AD-A246 079

(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DTIC
ELECTE
FEB 20 1992
S D
D

THESIS

NAVAL GUNFIRE SUPPORT: AN EXPANDABLE, OBJECT-ORIENTED, PROCESS-BASED SIMULATION

by

Richard L. Darden

September 1991

Thesis Advisor: Michael P. Bailey

Approved for public release; distribution is unlimited

92-03985

92 2 14 141

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL OR	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Including Security Classification)
NAVAL GUNFIRE SUPPORT: AN EXPANDABLE, OBJECT-ORIENTED, PROCESS-BASED SIMULATION

12 PERSONAL AUTHOR(S)
DARDEN, Richard L.

13 TYPE OF REPORT	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day)	15. Page Count
Master's thesis		1991, September	160

16. SUPPLEMENTAL NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Object-orientation, MODSIM, NGFS, Naval Gunfire Support, Programming, Process-based, Simulation, IBM PC

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis documents the design and implementation of a simulation of Naval Gunfire Support (NGFS) in a modern, object-oriented, process-based simualtion language called MODSIM II by CACI Corporation of La Jolla, CA. The main intent of the simulation is to build a model that will allow the Naval Weapons Support Center, of Crane, Indiana, to explore the effects of the individual component reliability of gun and shell components on the overall performance of the Naval Gunfire Support system. The choice of the language MODSIM II was made to evaluate the capabilities of an object-oriented, process-based simulation language. The model is an expansion of a similar model written in FORTRAN and the problems and solutions encountered in moving from that linear programming language to an object-oriented one are also documented. Additionally, the ease with which the simulation can be enhanced and upgraded is addressed, as the facility to do this is greatly affected by the model's object-orientation. Finally, the suitability of the use of a desktop computer, specifically an IBM PC compatible, as a platform for the development and the execution of large simulations is explored.

20 DISTRIBUTION/AVAILABILTIY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC	1a. REPORT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael P. Bailey	22b. TELEPHONE (Include Area Code) (408)646-2085 22c. OFFICE SYMBOL OR/Ba

DD Form 1473, JUN — Previous editions are obsolete. S/N 0102-LF-014-6603 — SECURITY CLASSIFICATION OF THIS PAGE Unclassified

Approved for public release; distribution is unlimited.

NAVAL GUNFIRE SUPPORT:
AN EXPANDABLE, OBJECT-ORIENTED, PROCESS-BASED SIMULATION

by

Richard L. Darden
Lieutenant, United States Navy
B.E.E., Georgia Institute of Technology, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

Author: _____
Richard L. Darden

Approved by: _____
Michael P. Bailey, Thesis Advisor

_____
Marcello Bartroli, Second Reader

_____
Peter Purdue, Chairman,
Department of Operations Research

ABSTRACT

This thesis documents the design and implementation of a simulation of Naval Gunfire Support (NGFS) in a modern, object-oriented, process-based simulation language called MODSIM II by CACI Corporation of La Jolla, CA. The main intent of the simulation is to build a model that will allow the Naval Weapons Support Center, of Crane, Indiana, to explore the effects of the individual component reliability of gun and shell components on the overall performance of the Naval Gunfire Support system. The choice of the language MODSIM II was made to evaluate the capabilities of an object-oriented, process-based simulation language. The model is an expansion of a similar model written in FORTRAN and the problems and solutions encountered in moving from that linear programming language to an object-oriented one are also documented. Additionally, the ease with which the simulation can be enhanced and upgraded is addressed, as the facility to do this is greatly affected by the model's object-orientation. Finally, the suitability of the use of a desktop computer, specifically an IBM PC compatible, as a platform for the development and the execution of large simulations is explored.

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

TABLE OF CONTENTS

I. INTRODUCTION 1

A. THE USERS 3

B. NAVAL GUNFIRE SUPPORT 3

C. THE ENVIRONMENT 3

D. SCOPE 4

II. NAVAL GUNFIRE SUPPORT 6

A. A TYPICAL NGFS EVOLUTION 6

1. ASSIGNMENT OF TARGETS 6

2. REGISTRATION FIRE 7

3. SPOTTING 7

4. FIRE FOR EFFECT 10

5. RESOLUTION 10

B. MEASURES OF EFFECTIVENESS 10

C. ADDITIONAL CONSIDERATIONS 11

1. COMPONENT FAILURE 11

(a) Interruptive Failure 12

(b) Non-Interruptive Failure 12

(c) Reassignment of Target 13

2. SHIP MOVEMENT 13

3. SIMPLIFYING ASSUMPTIONS 13

(a) Bivariate Normal Error 13

(b) Serial Failures 14

(c) Movement 14
(d) Damage Model 14
III. OBJECT-ORIENTED, PROCESSED-BASED SIMULATION 16
A. OBJECT-ORIENTED PROGRAMMING 16
1. DATA ENCAPSULATION 17
2. INHERITANCE 17
3. POLYMORPHISM 18
B. A PROCESS-BASED SIMULATION 18
C. MODSIM II 19
1. FEATURES 19
2. IMPLEMENTATION OF MODSIM II 20
(a) Object-orientation 21
(b) Simulation 21
IV. THE eF SIMULATION MODEL 23
A. OVERVIEW 23
B. SIMULATION EXECUTION 23
C. SIMULATION DESIGN 24
1. MODULES 24
(a) NGFS 24
(b) SIMCTRL 24
(c) HQ 25
(d) SHIP 26
(e) GUN 27
(f) TARGET 29
(g) SHELL 30
(h) SEED 30

(i) GLOBALS 31
(j) MIOMod 31
(k) MGrpMod 31
2. MODEL INPUT 31
3. MODEL OUTPUT 32
V. SIMULATION ANALYSIS 33
A. MODEL VERIFICATION 33
B. TERMINATION CONDITIONS 34
C. THE SIMULATION SCENARIO 34
D. THE BASE LINE SIMULATION RESULTS 35
E. ANALYSIS OF THE SIMULATION RESULTS 38
VI. PROBLEMS ENCOUNTERED 39
A. SWITCHING TO OBJECT-ORIENTED PROGRAMMING 39
B. TIMING AND STATE TRANSITION 39
C. MULTIPLE CONCURRENT PROCESSES 40
D. MODEL SIZE LIMITATIONS 41
VII. EXPANSION OF THE BASIC MODEL 43
A. PRIORITY OF TARGET ENGAGEMENTS 43
B. DAMAGE MODEL 44
C. RANDOM NUMBER SEEDS 44
VIII. CONCLUSIONS AND RECOMMENDATIONS 46
A. OBSERVATIONS 46
1. NGFS 46
2. OBJECT ORIENTED, PROCESS BASED SIMULATION 46
3. CONVERTING TO OBJECT-ORIENTED PROGRAMMING 46
4. LARGE SIMULATIONS AND THE PC 46

5. MODSIM II 47
B. SUGGESTIONS FOR FURTHER RESEARCH 47
C. CONCLUSIONS 48
APPENDIX A NGFS SIMULATION PROGRAM 50
APPENDIX B SAMPLE NGFS INPUT FILES 125
LIST OF REFERENCES 147
BIBLIOGRAPHY 148
INITIAL DISTRIBUTION LIST 149

LIST OF FIGURES

Figure 1. Direct Spot 8

Figure 2. Bracket Spot 9

Figure 3. Base Line Test Scenario 35

Figure 4. Base Line Test Scenario Results 37

ACKNOWLEDGEMENT

I would like to thank my lovely wife, Pamela, whose love, support, and understanding allowed me to complete this thesis.

I. INTRODUCTION

With the rapid advancement of technology, military weapon systems are becoming increasingly complex. Moreover, the weapon systems made feasible by these advancements in technology are extremely expensive. As the necessity for these advanced technology weapon systems meets with a declining federal budget, the military is faced with the challenge of finding a more cost effective means of designing and implementing new weapon systems. System simulation is one way in which the military might meet this challenge.

Simulation involves the use of computers to imitate the operations of the actual system. Simulation can be used in the design of weapon systems to determine if the systems will function as intended. Simulation can also be used in evaluation of weapon systems after they are built. This type of simulation often provides a more cost effective method of determining their optimum utilization as well as their effectiveness than actual tests. As the rising cost of technology faces the reality of a shrinking budget, simulation of military weapon systems will play an ever increasing role in the most efficient use of the military dollar.

Presently, most simulations are designed to model only a specific aspect of a system, such as the system's reliability performance. These simulations are written very specifically for the problem and cannot be readily adapted to different problems or to answer other questions that occur about the system being simulated or related systems. As a result, to address the different problems or answer further questions, totally new simulations must be developed that do not make much use of the effort or expense put into the first simulation. However, the advent of object-oriented, process-based

simulation languages such as MODSIM II by CACI Corporation of La Jolla, California [Ref. 1] promises to allow simulations that can be readily adapted to different problems and to answer further questions. The computer code generated also portends to be extremely reusable, thus allowing the effort and expense expended in generating code used in one simulation to be used with little or no change in a related simulation. This would greatly multiply the cost effectiveness of simulations and provide a high return on the investment of the military dollar.

This thesis documents the construction of a simulation of Naval Gunfire Support (NGFS) in MODSIM II. The model, called eF, seeks to simulate a Naval Gunfire Support Mission with various scenarios and different parameters to produce the time integral target value, the average mission completion time, and the average rate of fire for the guns involved as measures of effectiveness (MOEs). The choice of MODSIM II was made to evaluate the upgrade capabilities of an advanced object-oriented, process-based simulation language. The model is an outgrowth of a similar model written in the computer language FORTRAN by Michael P. Bailey, Marcelo Bartroli, Alexander Callahan, and Keebom Kang of the Naval Postgraduate School Faculty [Ref. 2]. This model of NGFS was developed for the Naval Weapons Support Center (NWSC), Crane, Indiana, to investigate the reliability of the individual components of the systems that make up Naval Gunfire Support. It is desirable to build a simulation that provides a foundation that not only can be used to study reliability, but also can be built upon or modified to study such varied topics as training effectiveness of naval gun crews or the most effective assignment of targets to ships.

This effort also addresses some of the problems encountered in moving from a discrete-event type of simulation in a non-object-oriented language to an object-oriented, process-based simulation. Finally, the ease at which the base simulation can be enhanced and expanded, owing to its object-oriented nature, will be explored.

A. THE USERS

The users of eF are specifically the personnel of the Naval Weapons Support Center of Crane, Indiana, which will use it to further their in-depth study of the reliability of major caliber naval ammunition and the effects of individual component reliability on overall NGFS system performance. This thesis will also be of general use to anyone who has struggled with expanding and maintaining discrete-event simulation or a simulation that has been written in a non-object-oriented programming language, by providing an example of what can be done given the tools generally available today and some of the benefits and pitfalls of so doing.

B. NAVAL GUNFIRE SUPPORT

The simulation presented in this thesis is one that simulates Naval Gunfire Support. A very brief definition of Naval Gunfire Support refers to guns on ships firing at targets in support of military operations. A more detailed description of Naval Gunfire Support follows in the next chapter.

C. THE ENVIRONMENT

The simulation program presented will run on an IBM compatible personal computer under DOS with both a hard disk and a floppy disk. For speed considerations, a fast (20MHZ+) 80386 computer is recommended along with a math coprocessor. However, neither the 80386 nor the math coprocessor is required to run the simulation. The simulation itself is written in MODSIM II, which is an advanced simulation programming language. The modification and compilation of the program requires the MODSIM II language and at least an IBM AT compatible (286+) computer with 4 megabytes of memory and a hard disk drive. In addition, the MODSIM II compiler requires either a Turbo C or Turbo C++ compiler by Borland International, Incorporated

of Scotts Valley, California [Ref. 3] since it compiles to the computer language 'C' which then is compiled by the 'C' language compiler to the native format of the personal computer. As stated in the conclusions, this configuration represents the minimum required to run the simulation. A larger model would require upgrading the system requirements to a personal computer running OS2 or a computer workstation.

D. SCOPE

The simulation described does not propose to be the definitive NGFS simulation program. Rather, the simulation is a general baseline simulation of Naval Gunfire Support that can be used as is to answer questions concerning the reliability of individual component parts and their effects on NGFS system performance. The impact of the object-oriented, process-based simulation programming language, MODSIM II, on the reliability, maintainability, expandability, and reusability of large simulations will be explored. Finally, some of the problems associated with moving from a discrete-event simulation in an older generation programming language to a process-based simulation in an object-oriented programming language will be discussed.

A word of caution is needed at this point to state that the simulation presented here is not a general purpose simulation. There is a danger inherent in general purpose simulations that was well stated by CAPT Wayne Hughes, noted author of *Fleet Tactics,* that is "General purpose models generally fail." [Ref. 4] This is because they attempt to do everything and succeed at nothing. A simulation should always be designed to address a specific problem. This design can build upon an existing simulation or reuse much of the existing simulation's code, but the existing simulation should not be used as is to answer questions for which it was not intended, without careful examination. This is because the assumptions, simplifications, and data used in the original simulation may not be applicable to the new simulation and may invalidate the results. The wheel should

not be reinvented each time it is needed, but it should be checked first to see that it fits the car on which it is to be used.

II. NAVAL GUNFIRE SUPPORT

Naval Gunfire Support is the use of major caliber naval guns against targets in support of military operations. An example of this would be a pre-assault bombardment of shore fortifications before an amphibious landing by Marines. Essential questions that need to be answered concerning NGFS are: How long does it take to destroy a given set of targets? What are the firing rates of the guns? How does component reliability of the gun and the shell affect the time required to destroy the target and the firing rate of the gun? What is the specific area that can be addressed by training that will yield the most significant improvement in NGFS? What is the best mix of shells and ships to destroy a target in the least amount of time? What is the most effective way to assign targets to ships to guns? These types of questions need to be addressed by a NGFS model. It is not the goal of this research to provide answers to all these questions, but rather to provide a basic simulation of NGFS that can be expanded and built upon to provide the answers to these types of questions and will directly answer the question of component reliability on the time necessary to destroy a target and the firing rate of the gun.

A. A TYPICAL NGFS EVOLUTION

A typical Naval Gunfire Support mission can be broken down into five basic parts: Assignment of Targets, Registration Fire, Spotting Fire, Fire For Effect, and Resolution. Each of these processes will be discussed below.

1. ASSIGNMENT OF TARGETS

The mission would commence with the order from the overall commander to the fleet commander to destroy the defensive installation. The fleet commander would

examine his assets and make the determination of which ships would attack which specific targets. A message would then be sent to the ship, tasking it with destroying the specific target(s).

2. REGISTRATION FIRE

The ship would receive the NGFS tasking order and then commence a registration fire of its weapons. This registration fire is an attempt by the ship to correct for the navigation system errors as well as the gun-system bias errors inherent in the ship. Several registration rounds are fired at an object outside the target area. These rounds are tracked to their landing point using the ship's radar. The data is then averaged and used as a bias to correct future firings. Registration fire is usually done once per day per ship. If the ship has more than one gun, only the lead gun (if available) completes a registration fire and the bias calculated is used for all subsequent firings by all guns aboard that ship.

3. SPOTTING

Once the registration fire is complete, the ship commences to spot the target. This consists of firing one shot at a time until the fall of the shot is zeroed in on the target. This is done so that once the gun is properly aimed it can commence to Fire For Effect (FFE) on the target at its maximum rate of fire without having to worry about the fall of the shot.

The basic process of spotting is that the fall of the shot is observed by a spotter who calculates a correction and communicates the correction back to the ship. The spotter can be either an observer on shore who can see the target, an observer offshore who can see the target, or a remotely piloted drone that can observe the target. The frame of reference for the observation is that the x direction always lies across the line of sight from the gun to the target and the y direction always lies in the line of sight from the gun to the target.

Once the spotter has communicated the correction back to the ship, the ship makes the necessary changes in bearing and azimuth to the gun and refires a spotting round. This continues until the spotting round meets the spotting criteria. The spotting criteria differs based on two types of spotting.

The first type of spotting is a Direct Spot. This is when the gun is aimed directly at the target. The successful completion criteria is that the shell falls within a certain radius, ***ECR***, of the target (See Figure 1). ***ECR*** is the estimated circular radius of the target and is derived from data in the Joint Munitions Effectiveness Manuals (JMEMS).

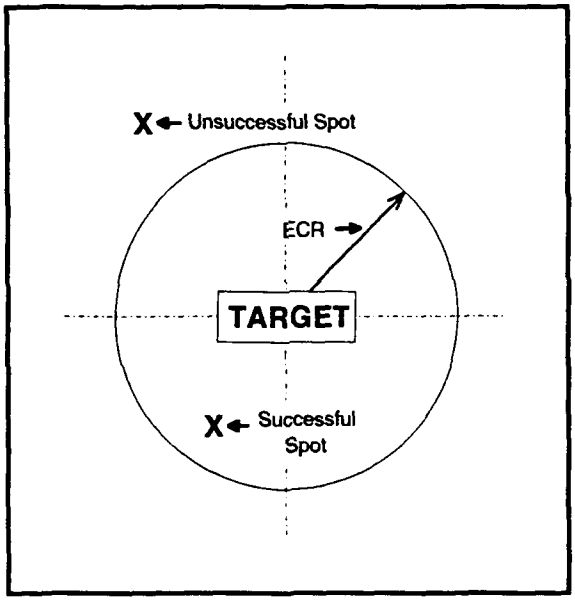

Figure 1. Direct Spot

The second type of spotting is a Bracket Spot. This type of spot fire seeks to improve the accuracy of the final aimpoint by bracketing the target with shells and taking the average of the two aimpoints used to achieve the bracket. The gun is initially aimed at the center of a long bracket box that is centered around a point that lies a distance ***ECR*ECRLong*** beyond the target (See Figure 2). ***ECRLong*** is a measure of the distance by which the bracket boxes are offset from the center of the target. It is a

dimensionless quantity that is multiplied by ***ECR*** to get the actual distance. The long bracket spot is successful when a shell is placed within the long bracket box, which has dimensions of ***ECR*BoxLong*** long and ***ECR*BoxWide*** wide. ***BoxLong*** and ***BoxWide*** are dimensionless qualities, like ***ECRLong***, that are multiplied by ***ECR*** to get the actual length and width of the bracket box. Once the long bracket spot fire has been successful, the process is repeated for the short bracket. The center of the short box is a distance **ECR*ECRLong** short of the target with the same dimensions as the long bracket box. Once a shell has been successfully placed in the short bracket box, the two aimpoints used to achieve the successful long and short bracket spots are averaged. This average aimpoint is used as the aimpoint for the fire for effect.

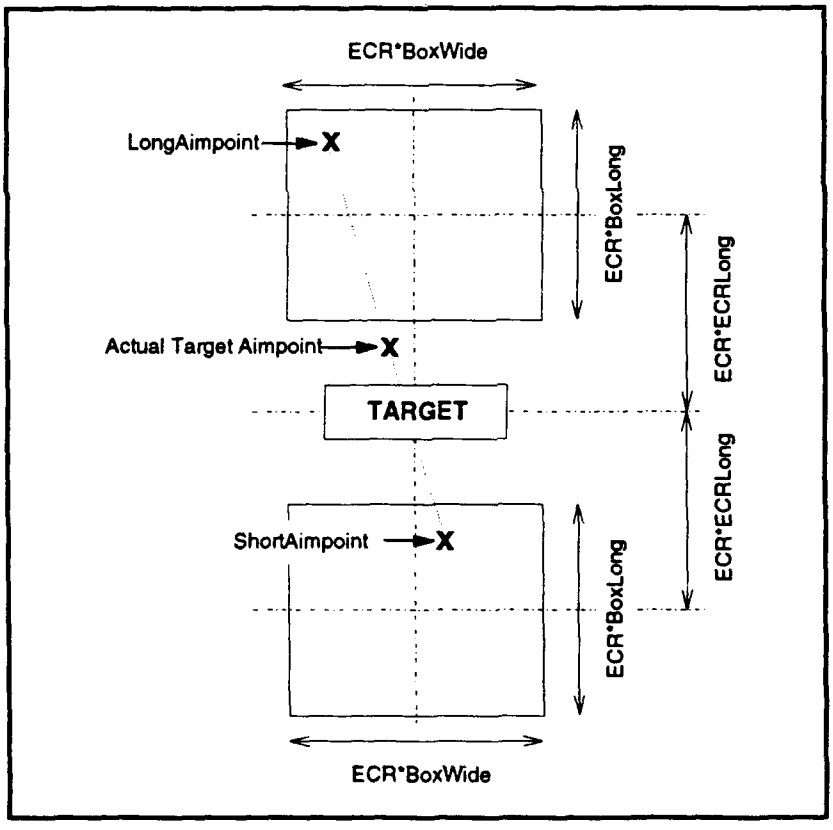

Figure 2. Bracket Spot

4. FIRE FOR EFFECT

Once the aimpoint for the gun has been established by spotting, firing for effect commences. Firing for effect consists of firing a set number, ***RndsPerFFE***, of the appropriate type of shell (High Explosive, Armor Piercing, Submunition, etc.) at the target at the maximum rate of fire of the gun. Once the fire for effect is complete, the spotter performs a battle damage assessment of the target.

5. RESOLUTION

In the resolution phase, the ship receives the results of the FFE from the spotter and acts on them. If the target is destroyed, the ship communicates this to the headquarters and another target is assigned to the ship if an unassigned target is still available. If the target was damaged but not destroyed by the FFE, another FFE is commenced on the target using the same aimpoint. If the target was not damaged by any FFE round (ie. it was missed), a new round of spotting occurs on the target to establish a new aimpoint for a subsequent FFE. This process of spotting followed by fire for effects is continued until the target is destroyed.

B. MEASURES OF EFFECTIVENESS

The primary MOE that will be used to evaluate the NGFS system will be the time integral target value. This is the integral over time, of a target's ***LifePoints***, summed over all targets. A target's ***LifePoints*** are a measure of its ability to sustain damage and still operate and are initially set to some fixed value depending on the type of target. They are adjusted downward according to the amount of damage a target receives. A ***LifePoints*** value of zero would correspond to a target that has been destroyed. This MOE is used because it gives a better measure of the rate at which targets are damaged than just the time required to destroy all the targets. The smaller the time integral target

value the faster the targets are damaged and ultimately destroyed. The problem with this MOE is that it has no ready physical interpretation.

The solution to this is to also provide the secondary MOE's of the mission time and the firing rates of the guns. The mission time is the time required to destroy all the targets. The firing rate of each gun is the average firing rate of that gun over the time that there exists a target for that gun to fire at. These MOEs have the obvious physical interpretations and serve as a check on, and to enhance the interpretation of, the primary MOE of time integral target value. These MOEs just do not provide as much information as the primary MOE.

The use of these MOEs will allow the determination of the minimum required reliability levels of round and gun components to achieve a given effectiveness against the target. This was the original purpose for the FORTRAN based simulation of NGFS [Ref. 2].

C. ADDITIONAL CONSIDERATIONS

In addition to the basic NGFS scenario described, there are some other factors that affect the performance of the guns; the main factor, and the one with which NWSC is concerned, is component failure. Ship movement while firing also affects the rate of fire and will be considered.

1. COMPONENT FAILURE

Each of the mechanical components involved in a Naval Gunfire Support mission has associated with it a certain degree of reliability. A shell, for example, consists of fuses, primers, propellant, projectile, and explosive. All of these components of the shell and more components of the gun can fail and prevent the shell from landing on target, detonating, and causing damage to the target. Fortunately, the components basically operate in a serial chain. There is an electrical signal to fire that is amplified

and sent to the primer. The primer ignites and causes the propellant to burn. The propellant burns and the hot gases produced push the shell out of the barrel. The shell then flies on a ballistic trajectory to the target. When the shell arrives at the target, the fuse arms the shell and starts a sequence of ever increasing explosive charges that results in the detonation of the shell. (see Weapon Systems Fundamentals, NAVORD OP 3000 VOLUME 2, First Revision, for details.) As will be seen, the serial nature of this chain greatly affects the failure analysis of the system.

Since the system is serial in nature, the failure of a component in the first part of the chain will prevent a failure in any components further down the chain. For example, if the primer fails, the propellant will not ignite, the shell will not be ejected from the gun, the shell will not fly to the target, and the fuse will not arm the shell and cause detonation of the explosive. When a failure occurs anywhere in the chain, a round is deemed to be non-effective and will not cause damage to a target. When a failure does occur, it will be either an interruptive failure or a non-interruptive failure, based on its impact on subsequent firings.

(a) Interruptive Failure

An interruptive failure is a failure that causes the gun to stop firing for a certain amount of time. An example of this type of failure would be a failure of a propellant charge. Since the propellant did not ignite, the shell is still in the barrel of the gun in some unknown state and must be cleared before firing can resume. This takes a set amount of time to be done correctly. This interrupts the firing process and is therefore considered an interruptive failure.

(b) Non-Interruptive Failure

A non-interruptive failure is a type that occurs if the shell successfully leaves the gun but fails to cause damage to the target. Examples of a non-interruptive failure include failure of the fuse, failure of the explosive, guidance errors, and reduced

initial velocity. The key factor in all these types of failures is that there is no impact on subsequent gun firings. In other words, they are non-interruptive and the gun can fire its next shot with no delay.

(c) Reassignment of Target

In addition to the two types of failures, there is a further classification of interruptive failures. This classification depends on the length of time that the gun will be out of action. If it is out of action for a time that exceeds a certain value, ***TooLong***, the target is reassigned to another gun on that ship, if one is available, or released back to headquarters to be reassigned if an idle gun is not available on that ship.

2. SHIP MOVEMENT

Ship movement affects the firing rate of the gun in that the distance the ship (gun) is from the target affects the flying time of the shell. Also, during the time when a ship is maneuvering, the guns cannot be fired accurately, so fire is suspended during the maneuver. As a result, the firing rate is less than with a stationary ship. Since this has a direct impact on the firing rate of the guns and the target destruction times, the movement of ships will be considered.

3. SIMPLIFYING ASSUMPTIONS

There were several key assumptions made in the design of the model to simplify its construction. The key assumptions were bivariate normality, serial failures and movement assumptions.

(a) Bivariate Normal Error

All accuracy errors in the system are assumed to be bivariate normal. This applies to the navigation system accuracy of the ship, to the aimpoint accuracy of the gun, and the actual fall of the shot.

(b) Serial Failures

All failures of the gun and shell will be considered to be serial failures. That is a failure of a component that will prevent any failure of a component further down the line from occurring.

(c) Movement

Normal movement of a ship will be on a straight course at a set speed. Changes from the base course and speed will be by maneuvers that will occur according to an input track. Firing will not occur during a maneuver. A request for a maneuver will only be postponed for one complete round of an FFE. A request for a maneuver will interrupt any other event and cause it to recommence once the maneuver is complete. The result of the maneuver will be to reposition the ship offset by advance and transfer on a new course and speed at a new time equal to the time at the start of the maneuver plus the duration time of maneuver.

(d) Damage Model

The amount of damage a target sustains due to the impact of a shell is dependent on several factors, such as the type and size of the shell, the miss distance, and the type and size of the target. It was desired to incorporate a damage model that would take into account all these factors in assessing the damage a target sustains. In order to do so, the following variables are defined. Each target is assigned a positive real value, ***LifePoints***, as a measure of its ability to sustain damage and still operate. The higher the ***LifePoints***, the more damage a target can sustain. A target's ***LifePoints*** is initially set to a positive value and is decreased by the damage a target sustains. When a target's ***LifePoints*** reach zero the target is assumed to be destroyed. Each type of shell has an associated damage factor, ***DF***, for a particular target. The ***DFs*** are [0.0, 1.0] real variables that are maintained by the target and represent a measure of the target's

susceptibility to damage from a particular type of shell. The ***ECR***, or estimated circular radius, is a positive real variable that is a measure of the size of a target.

Each shell has ***Type*** associated with it. The types are user definable and standard types would include high explosive (***HE***), fragmenting (***FRAG***), armor piercing (***AP***), and submunitions (***SUB***). ***MaxDamage*** is a positive real variable that is a measure of the shell's destructive power. ***EDR***, estimated damage radius, is positive real variable that is a measure of the effective killing radius of the shell.

Given these variable definitions, the formula for calculating the damage a particular target sustains due to the impact of a particular shell is the following:

$$Damage = MaxDamage * DF * \left[\frac{DamageRadius - MissDistance}{DamageRadius} \right]^2$$

where

DamageRadius = ***ECR*** + ***EDR***

MissDistance = The distance the shell misses the center of the target by.

DF = Damage factor for this type of shell for this particular target.

Damage = 0.0 if ***MissDistance*** $\geq$ ***DamageRadius***

This formula gives sufficient latitude to account for the major variables in predicting target damage. However, the formula has not been validated and will presently only give relative results based on estimates for the parameter values. The model would have to be validated, then calibrated, using actual data, before the results would have any true physical significance in terms of the amount of damage a target sustains.

III. OBJECT-ORIENTED, PROCESSED-BASED SIMULATION

Since a goal of this research is to explore the impact of object-oriented, process-based simulation on current simulations, it is necessary to introduce some of the basic concepts associated with each of these ideas. Also, since MODSIM II is the language in which **eF** is written, its particular implementations of the basic concepts of object-oriented, process-based simulation will be covered.

A. OBJECT-ORIENTED PROGRAMMING

Object-oriented programming (OOP) is an old programming concept that has seen a resurgence of popularity through the release of modern OOP languages [Ref. 5]. The resurgence in popularity stems from the promise of object-oriented languages to allow the writing of code that is more reliable, expandable and reusable than code written in other types of programming languages. The fundamental concept of OOP is that the code is structured around objects and interactions between those objects much like in the real world. The objects can either be abstract objects or correspond to actual physical objects. The objects have data (fields) and procedures (methods) that act upon their data. They communicate with other objects by way of sending messages to them. In addition, all OOP languages have the characteristics of being strongly typed and block structured, much like Pascal or C. Since these concepts are common among current high level programming languages, they will not be addressed. Instead, the concepts of data encapsulation, inheritance, and polymorphism that distinguish OOP will be explored.

1. DATA ENCAPSULATION

Data encapsulation is the concept by which an object maintains its own data in an orderly and structured fashion and allows only selective access to its data by any object other than itself. Data encapsulation can also be called data hiding since any attempt to access an object's data except through specifically declared methods is not allowed.

Data encapsulation promotes the expandability of a program since the interface to the data is fixed outside of an object. This means that if something is changed inside of an object, it has no effect on the routines calling the object as long as the object's method of calling does not change.

2. INHERITANCE

To understand the concept of inheritance, it is necessary to understand the concept of classes, subclasses and instances. "...a class is a set of closely related objects sharing similar attributes." [Ref. 6:p. 5] A subclass is a class that has all the characteristics of its parent class and can add new characteristics of its own. An instance is a particular realization of a member of a class.

An example of a class would be moving vehicles. A subclass of moving vehicles could be automobiles. An instance of an automobile class would be a Ford Taurus. An example of inheritance is that an automobile derives some of its attributes from the moving vehicle class. In other words it has a name, a current position, speed and direction and can be told to go, stop and turn. To these attributes it adds a manufacturer, four wheels, a type, a steering wheel and the ability to start the engine, stop the engine, open door, etc. The Ford Taurus is an instance of an automobile that fills in the name attribute inherited from the moving vehicle class with Taurus. It fills the manufacturer field with Ford. It also knows what to do to when told to go, stop and turn since it inherited those methods from the moving vehicle class.

Inheritance promotes reusability since code is reused (inherited) from a class to a subclass. It also allows for a change to be effected in just the parent class and inherited downward to all subclasses. This greatly simplifies the programming process.

3. POLYMORPHISM

Polymorphism is the concept of overloading of operator names. This concept allows the same operator (method name) to be implemented differently among different objects. In **eF** for example, both a ship object and a gun object can be told to ***EngageTarget*** and be passed a target to engage. How the two objects respond to the operator is completely different. The ship determines an idle gun and tells it to ***EngageTarget***; the gun actually starts the process by which it will spot the target and fire for effect on the target until the target is destroyed.

Polymorphism enhances the structured aspect of the code as well as reducing its bulkiness as it allows programmers to tell an object to do something like telling a moving object to go without having to worry about whether the object walks, swims, or flies.

B. A PROCESS-BASED SIMULATION

A process-based simulation is a simulation environment that allows for time to pass in an event, or method. This is compared to a discrete-event simulation in which time can only pass between events. George S. Fishman describes process-based simulation, or as he calls it process interaction approach, in the following manner:

> ...the process interaction approach provides a *process* for each entity in a system. Each temporary entity moves through the system and consequently through time. Occasionally, a temporary entity encounters an impediment to progress and must *wait*. [Ref. 7:p. 139]

The main benefit of process-based simulation is that it allows simplifications in larger models where it is hard to follow the flow of logic that defines the behavior of an

object. Furthermore, multiple, concurrent processes can be occurring for the same class of object or even the same object. Finally, processes can interact with each other and cause each others' behavior to change. All these features of a process-based simulation come in addition to the normal features of a discrete-event simulation as time does not have to pass in a method, or event, and can pass between methods, or events. [Ref. 8:p. 137]

The major disadvantage of process-based simulation as compared to discrete-event simulation is also pointed out by Fishman and is that in a process-based simulation there is less program control [Ref. 7:pp. 138-139]. This can lead to problems in timing and state transition, as can be seen in Chapter VI, but with careful attention to the program design, the advantages of process-based simulation far outweigh the disadvantages.

C. MODSIM II

MODSIM II is an object-oriented programming language that is modular, block structured, and provides support for discrete-event simulation. It is loosely based on the language Modula-2. It was designed to bring the best features of contemporary programming languages to bear on real world simulation problems. The following are some of the features and particular implementations found in MODSIM II as described in the reference manual for MODSIM II [Ref 8].

1. FEATURES

One of the key features of MODSIM II is its portability. The MODSIM code is compiled to C code which is then compiled by the machine's native C code compiler to an executable file to run on that machine. The result is that the MODSIM code can be transferred from a PC to a workstation, recompiled, and run with no changes to the MODSIM code. This is a very valuable feature which allows initial model development

work to be done on a PC and later, when the model grows too big to be run on the PC, it can be transferred to a workstation or mainframe with no effort lost.

MODSIM II is modular. Different parts of a program can exist in separate files that can be edited and compiled separately. A single module can also be shared by multiple programs. This facilitates the reuse of code.

MODSIM II is strongly typed. All expressions and assignments are checked by the compiler for consistency. This eliminates cross type assignment errors that can be difficult to track down. It also allows for user defined data types, which makes for a much simpler, readable program.

MODSIM II is block-structured. A block is made up of definitions and executable statements. Blocks can be nested. They also limit the scope of declarations to be contained inside the blocks.

MODSIM II is object-oriented. It contains all the key features of OOP: data encapsulation, inheritance, and polymorphism. The implementation of these features is straight forward and greatly enhances the utility of the language.

MODSIM II is simulation capable. The simulation capabilities are provided through the use of library modules. These library modules implement all the necessary bookkeeping procedures to allow discrete-event simulation. In addition, MODSIM II allows for process methods to be used which can elapse simulation time (process-based simulation).

2. IMPLEMENTATION OF MODSIM II

The implementation of MODSIM II is fairly straight forward and anyone familiar with a modern modular, block-structured, strongly typed language such as Pascal or ADA should have very little trouble understanding MODSIM II code. For that reason, those features of MODSIM II will not be further discussed. The implementation of the object-oriented features of MODSIM II are similarly straight forward. However,

due to the relative newness of OOP, these features will be further discussed along with the implementation of simulation in MODSIM II.

(a) Object-orientation

An object in MODSIM II is defined as having data, referred to as fields, and procedures, referred to as methods. The fields can either be public, in which they can be accessed by other objects for read only, or private, in which they cannot be accessed by any other object. In any case, the fields of an object can only be modified by the object's own methods. There are two types of methods that an object can have: an ASK method and a TELL method. An ASK method is similar to a procedure found in other languages and can have both input and output arguments. It also cannot elapse simulation time. A TELL method can only have input arguments and can elapse simulation time. The invocation of both these types of methods is in the format of:

ASK (TELL) object TO method name IN time duration

The IN time duration clause is optional and allows for direct discrete-event simulation modeling.

Objects can inherit both fields and methods from other types of objects. An object can also override an inherited method and substitute its own implementation of that method in place of the inherited implementation.

(b) Simulation

The simulation capabilities of MODSIM II are built into a module called ***SimMod***. They consist of a ***StartSimulation*** procedure that starts a simulation, ***SimTime()***, that returns the current value of simulation time, and some other procedures that allow synchronization of events through trigger objects and interruption of time elapsing processes.

The key method by which an object interfaces with the simulation module is through the use of a WAIT statement. A WAIT statement elapses simulation

time in a TELL method. The wait can be for a certain length of time (WAIT DURATION) or for an object to complete an action (WAIT FOR object TO {TELL method}). The key point is that only TELL methods can contain WAIT statements. In addition each WAIT construct has an optional ON INTERRUPT clause which is executed when the particular process of an object instance is interrupted. This allows a process that is dormant in a wait condition to be awakened before the completion of the WAIT condition.

IV. THE eF SIMULATION MODEL

The **eF** model is a basic simulation of NGFS tailored to answer specific questions about gun performance, as measured by the time integral target value, firing rate, or time to destroy targets, as a function of the reliability of gun and shell components. It is written in MODSIM II and designed to provide a foundation which can be expanded and reused to answer further questions about NGFS.

A. OVERVIEW

The simulation program consists of 19 MODSIM II modules consisting of one main module and nine paired definition and implementation modules. The main module is the name of the executable file and is called NGFS. The file naming convention used by MODSIM is that all MODSIM files will end in .MOD and be prefixed by an "M" if it is the main module, a "D" if it is a definition module, and an "I" if it is an implementation module. A main module contains the main program and is the name of the executable file created. A definition module contains type and variable definitions that can be exported to other modules. An implementation module contains the actual code to implement the definition module. Each module is described in detail later on in this chapter.

B. SIMULATION EXECUTION

Execution of the simulation is straightforward. The user uses an ASCII text editor to set up the input files, ***SIMPARM.DAT*** and ***ACTUAL.SCN***. ***SIMPARM.DAT*** contains the simulation parameters required for the simulation. ***ACTUAL.SCN*** contains the

scenario file used to build the simulation experiment. Next he runs the program by typing NGFS. The program echoes all its inputs to a file called ***INPUT.LOG***. The results are presented in a file called ***NGFS.OUT***.

C. SIMULATION DESIGN

The design of the simulation is based on objects that correspond to their real world counterparts. There is a ***HQ*** object that corresponds to a NGFS headquarters, a ***Ship*** class object from which ships are built, a ***Gun*** class object, a ***Target*** class object and a ***Shell*** class object. Each of these objects is implemented as a separate module, consisting of a definition module and implementation module pair. These modules and the utility modules ***SIMCTRL, GLOBALS, SEED, MIOMod,*** and ***MGrpMod*** are described next.

1. MODULES

(a) NGFS

The NGFS main program module is contained in the file ***MNGFS.MOD***. It is a very simple main program that only outputs the date and the time and calls the ***RunSimulation*** procedure contained in ***SIMCTRL*** module.

(b) SIMCTRL

This module is contained in the files ***DSIMCTRL.MOD*** and ***ISIMCTRL.MOD***. It consists of only one procedure, called ***RunSimulation.*** This procedure is designed to execute the simulation and will eventually be called from a larger **eF** control program to do just that. The basic flow of the simulation is that the procedure creates a new ***HQ*** object and tells it to ***CreateScenario***. It then enters a loop consisting of telling the ***HQ*** to ***InitializeScenario*** and ***RegisterAllShips***, followed by ***StartSimulation***. When the mission is complete, ***HQ*** is told to ***CalculateRunStats.*** ***CalculateRunStats*** returns the boolean variable, ***SimDone,*** which is true when the

simulation meets the stopping criteria. This loop is repeated until ***SimDone*** is true. ***HQ*** is then told to ***ReportStats*** and the procedure terminates.

(c) HQ

The ***HQ*** module consists of the files ***DHQ.MOD*** and ***IHQ.MOD.*** It defines the ***HQObj*** (headquarters object) type and supporting constructs. The headquarters object acts as the controlling object for the simulation. It maintains as its fields the global MOE values. It also maintains the current simulation stopping method and associated values. It maintains a ranked queue of targets and an unranked queue of ships. A ranked queue is a collection of references to objects maintained in rank order based upon some arbitrary ranking order. This facility is provided by MODSIM II as a ***RankedObj***. An unranked queue is the same as a ranked queue but is maintained in first in, last out order, and is provided by MODSIM II as a ***QueueObj***.

The ***HQ*** object first sets up the simulation when it is told to ***CreateScenario.*** It does this by reading in the initial simulation parameter data from the file ***SimParm.DAT***. It then creates the scenario by reading in the data from the ***Actual.SCN*** scenario file. Based on the input data, the ***HQ*** object creates the ships and targets needed for the scenario.

Next, for each repetition, it is told to ***InitializeScenario*** in which it resets the scenario to run another repetition. This is followed by being told to ***RegisterAllShips***. This method tells all the ships in the scenario to ***RegistrationFire.*** The ***RunSimulation*** procedure then starts the repetition by issuing the ***StartSimulation*** command. When the repetition is complete, the ***HQ*** object calculates the statistics for that repetition when told to ***CalculateRunStats***. It also returns whether or not the simulation is complete as the boolean variable ***SimDone.*** When the simulation is complete, the ***HQ*** object reports the final statistics when told to ***ReportStats.***

When a repetition is running, the ***HQ*** object is kept informed of the progress of the simulation run by the methods, ***UpdateTargetValue*** and ***AssignTargets.*** ***UpdateTargetValue*** is a method that calculates the new total target value for all targets and allows this value to be integrated over time into the time integral target value.

AssignTargets is a TELL method used to tell the ***HQ*** object to go down the list of targets and assign any target that is not destroyed and not presently assigned to the next ship that has a gun available. This method is called by a ship when its engagement status changes or when it has completed its registration fire. It is also called by a ship when a gun failure forces the ship to release the target.

(d) SHIP

The ship module consists of the files ***DShip.MOD*** and ***IShip.MOD.*** These modules define and implement a ***ShipObj*** and related types. A ***ShipObj*** is an object that simulates a ship in the NGFS scenario. It is a subclass of an ***IDObj*** which is defined in the module ***GLOBALS***. It consists of three basic groups of methods. The first creates and initializes the Ship object. The second group handles the maneuvering of the Ship object. The final group handles the engagement of targets by the Ship object. It also contains many fields that allow it to keep track of its status. It maintains a list of its guns in the queue, ***GunList.***

The ***ShipObj*** is created by the ***HQ*** object which immediately asks it to **CreateShip** and passes it the ship type, ID, and initial course, speed, position, and track. The ship in turn initializes its fields based on the type of ship it is by reading the appropriate ship data file (*.***SHP**). The ***ShipObj*** in turn creates gun objects that will simulate its guns. It also sets up its initial course, speed, and position and reads in the track the ship is to follow from the maneuver data file (*.***MAN**). The ***ShipObj*** is initialized prior to each simulation run by the method ***InitializeShip.***

The method ***UpdatePosition*** updates the position of the ship based on its present course and speed from the last position. The methods ***Maneuver*** and ***RecManvGranted*** are used to maneuver the ship. The initial instance of the ***Maneuver*** method is scheduled by ***InitializeShip***. Subsequently, the maneuver method requests a maneuver from all the ships guns at the appropriate time. When all the guns report that they are ready for a maneuver by calling the ship's ***RecManvGranted*** method, a maneuver occurs. Simulation time is allowed to advance for a time equal to the duration time (***DTime***) of the maneuver. Then the ship's position is then offset by the maneuver advance and transfer. The ship then releases its maneuver request by telling all its guns to ***CancelManeuver.*** It then schedules the next maneuver in by telling itself to ***Maneuver*** and passing itself the pointer to the next maneuver record. The method ***HaltManeuvers*** is called by the ***HQ*** object to stop the ship from maneuvering once the mission is completed.

The ship is initially told to ***RegistrationFire.*** This causes the ship to choose its first available gun and tell it to ***RegistrationFire.*** When the gun completes its registration, it returns the results to the ship by telling the ship to ***ReceiveRegStatus.*** If the registration was successful, the ship tells itself to ***UpdateEngagementStatus*** which updates the ship's ***EngagementStatus*** field and tells the ***HQ*** object to ***AssignTargets*** if the ***ReAssignTargets*** argument was true. If the ***RegistrationFire*** was unsuccessful, the ship tells itself to ***RegistrationFire*** again. If no guns are available during an attempt to ***RegistrationFire,*** the ship waits until a gun is repaired then tries again. A ship is told to ***EngageTarget*** and passed a target to engage by ***HQ*** object or on occasions by itself. In response, the ship tells the next available gun to ***EngageTarget.***

(e) GUN

The gun module consists of the files ***DGUN.MOD*** and ***IGUN.MOD.*** This is the most complex module. It defines a ***GunObj*** and supporting types. A ***GunObj***

corresponds to a gun onboard a ship and the associated fire control and communications facilities. A ***GunObj*** is also a subclass of ***IDObj*** and is created by a ***ShipObj.*** The ***ShipObj*** then tells it to ***CreateGun*** and passes it the type of gun it is to be. The ***GunObj*** then reads the ****.GUN*** file for the type of gun. The ****.GUN*** file contains the information defining the characteristics of the particular type of gun. One of the characteristics is the type of shells available to that gun. To save having redundant information, only one copy of each type of ***ShellObj*** is created for the entire scenario, no matter how many guns use that type of shell. The original types of ***ShellObj*** are kept in the queue, ***ShellList,*** defined in the module ***GLOBALS***. References to these ***ShellObj***'s are kept in the ***Magazine*** for each gun. This is a linked list of type ***MagazineTYPE*** that contains the type of shell, the pointer to the original copy of the shell in the ***ShellList,*** and a pointer to the next magazine record. Once the ***GunObj*** has been created, it is initialized before each repetition by calling ***InitializeGun***.

The ***GunObj*** has many fields that allow its status to be tracked. Some of the main ones are its ***Status*** and ***Process*** fields, that act in the same manner as the fields of the same names for the ***ShipObj***. ***HOT*** is a boolean variable that is set to true when the gun has fired more than ***ShotsBeforeHot*** rounds during a single simulation. When ***HOT*** is set, all repair times are taken to be their ***MTTRH***, or mean time to repair-hot, values vice their ***MTTRC***, or mean time to repair-cold, values.

The gun responds to a ***RequestManeuver*** from its ship by granting ***ClearedToManeuver*** and telling the ship to ***RecManvGranted*** when the gun is a a stopping point in its operation. ***CancelManeuver*** causes the gun to resume the process it was doing when the maneuver was requested.

A gun can be told to ***RegistrationFire***. This causes the gun to fire ***NumRegRounds*** worth of rounds at a simulated reference target, located ***RegRange*** away. The shell is told to ***FlyToTarget*** as normal, but in this case the target is a

NILOBJ and the shell tells the ship to ***TrackRound*** when it has finished its flight. This causes the ship to notice where the shell lands. When all the registration rounds have been fired, the ship calculates its ***Bias*** which is the correction for internal navigation and gun systems errors based on the average of the registration rounds.

When told to ***EngageTarget,*** the gun checks to see what kind of spotting regime the target requires, then tells itself to ***SpotTarget***. This causes the gun to ***Fire*** at the target until the spotting criteria is met. The ***Fire*** method is a general purpose method used by higher level routines to simulate the actual firing of the gun at the designated target. It also handles gun and shell failures. A gun or shell component can fail in three ways: a ***Misfire*** (the firing cycle of the gun is unaffected), a ***BrokeSoft*** (the firing cycle is interrupted for a time less than ***TooLong*** and does not result in the release of the target), or a ***BrokeHard*** (the firing cycle is interrupted for a long enough time to cause the target to be reassigned to another gun on the same ship, if one is available, or to another ship).

Once the target is successfully spotted, the target begins its FFE. At the completion of the FFE, the target reports its status to the gun and the gun will either fire another round of FFE at the target, respot the target, or ask for another target since its present one is destroyed.

(f) TARGET

The ***Target*** module is contained in the files ***DTARGET.MOD*** and ***ITARGET.MOD***. It implements a ***TargetObj*** and supporting types. The target is created by the ***HQ*** object. It is immediately told to ***CreateTarget*** and passed its creation parameters. The target object then reads the rest of its parameters in from the ****.TGT*** file associated with the type of target. This object is also initialized before every run by calling the ***InitializeTarget*** method. The target can be told to ***StandbyForSpot*** or ***StandbyForFFE***. These methods alert it to an incoming shell and tell it how to evaluate

the impact. The method ***ImpactRound*** is used to explode the shell and calculate any damage done to the target. The gun object then waits for the round to impact then requests the target to ***ReportSpotResult*** or ***ReportBDA***. The target responds to these request by informing the gun of the appropriate information.

(g) SHELL

The ***Shell*** module consists of the files ***DSHELL.MOD*** and ***ISHELL.MOD***. These files implement the object ***ShellObj***, and the procedure ***RangeFunction***. The shell object implements the shell in the NGFS scenario. It is created once per scenario by the first gun to use that type of shell. The gun then tells the shell to ***CreateShell*** and the shell reads its parameters from the appropriate ***.SHL** file. The shell is then placed in the global variable, ***ShellList***. When a gun needs to fire a shell, it first clones the appropriate type then tells it to ***FlyToTarget*** and gives it a target to which to fly. The shell delays the appropriate flight time and then tells the target to ***ImpactRound***.

This module also contains the procedure ***RangeFunction*** that allows the variance of the aimpoint of the gun and the impact point of the shell to be a function of range. The formula used is: $\sigma(r) = \sigma * r * RangeFactor$. A value of minus one for the ***RangeFactor*** is used as a flag and will result in $\sigma(r) = \sigma$.

(h) SEED

This module consists of the files ***DSEED.MOD*** and ***ISEED.MOD***. These files define a ***MRandObj***, a ***SeederObj***, and a ***seeder***. The ***MRandObj*** object is a modification to the basic ***RandObj*** that adds a method called ***GetSeed***. This method gets the next random number seed from the appropriate list in the ***seeder***. The ***SeederObj*** is an object type for the object instance, ***seeder***, also defined in this module. The ***seeder*** is an object that reads in several lists of random number seed strings from the file

NGFSSEED.DAT. It then returns the next seed available in the specified seed string in response to a *GetNextSeed* call.

(i) GLOBALS

The module *GLOBALS* is contained in the files *DGLOBALS.MOD* and *IGLOBALS.MOD*. This module defines global types and variable used in the simulation.

(j) MIOMod

The module *MIOMod* is contained in the files *DMIOMod* and *IMIOMod.MOD*. This module defines a *MStreamObj* that contains utility methods that allow for easier input and output. They also provide methods for automatically logging read input to the *MStreamObj*, *InputLog*, also defined in this module.

(k) MGrpMod

The module *MGrpMod* is contained in the files *DMGrpMod.MOD* and *IMGrpMod.MOD*. This module modifies the basic queue object to form a *ListObj* that allows a query of a pointer to an object based on a field *ID* of an *IDObj*. The module defines a *ComponentObj* and a *ComponentListObj* which implement the method by which Components of a Gun or a Shell are kept track of and queried for failures.

2. MODEL INPUT

The simulation parameters are read in from the file, *SIMPARM.DAT*. The scenario data, what targets and ships to create, is read in from the file, *ACTUAL.SCN*. The data that defines the individual ships, guns, shells, and targets are read in from the files, **.SHP*, **.GUN*, **.SHL*, and **.TGT*, respectively. The data for these files is currently fictitious data, that represents the actual data that is presently being collected and built into a data base. The maneuver tracks for the ships are kept in **.MAN* files. The random number seeds for the simulation are kept in the file *NGFSSEED.DAT*.

3. MODEL OUTPUT

All input values that the model uses are logged to the ***INPUT.LOG*** file for every run to ensure that a record of the input that causes a particular output is always available. A flag can be set in ***SIMPARM.DAT*** that will cause a logging of all major events in each simulation repetition to be logged to the file ***EVENT.LOG***. Due to the large number of events that occur in each repetition, it is not recommended that the events be logged for more than two to three repetitions.

The statistical output of the simulation consists of the gun firing rates for each gun in the system, the overall average firing rate, the mission time and the time integral target value for each run. In addition, at the termination of the simulation, the average values of the firing rates, mission time, and time integral target value, along with their confidence intervals, are all written to the output file called ***NGFS.OUT***.

V. SIMULATION ANALYSIS

This simulation is a terminating simulation [Ref. 9:p. 280]. The mission will end at the time when all the targets are destroyed. Let μ be the expected value of the random variable X which is one of the MOEs. Then, if $X_1, X_2, ..., X_n$ are estimators of X assumed to be independent and identically distributed (IID), then for large n, by the *Central Limit Theorem*, the 100(1-α) percent confidence interval for μ is given by:

$$\mu = \overline{X}(n) \pm z_{1-\alpha/2}\sqrt{\frac{S^2(n)}{n}} \qquad \text{[Ref. 9:p. 149]}$$

where $S^2(n)$ is the sample variance, n is the number of samples, $z_{1-\alpha/2}$ is defined by the equation $1-\alpha/2 = P(Y < z_{1-\alpha/2})$, where Y is a standard normal random variable and $0 < \alpha < 1$ is the significance level. For this simulation the normal significance level will be $\alpha = 0.05$, but this is user selectable in the ***SimParm.Dat*** input file as the ***InvNormalCI***, which is the value of $z_{1-\alpha/2}$ for the desired 100*(1-α/2) percent confidence level. This is the basis for establishing the confidence intervals around the average value of the various MOEs.

The statistics are calculated using MODSIM II-provided statistical objects that automatically collect the mean and population standard deviation for a given variable. The value for the population standard deviation is then converted to the sample variance to obtain the confidence intervals.

A. MODEL VERIFICATION

The model has been verified to work correctly against a number of test scenarios. The base line test scenario runs and the output has been examined and is believed to be

correct. The model has not been validated as this would require comparing its results to results obtained from actual ship firings. Since it has not been validated, it has also not been calibrated and the results obtained should only be viewed relative to one another and not as absolute numbers. It is the intention to validate the model using data obtained from actual range firings in the future.

B. TERMINATION CONDITIONS

The simulation is terminated by one of two methods. If the variable ***StopMode*** is set equal to ***NumReps***, then the simulation terminates after ***MaxReps*** have been completed. If ***StopMode*** is set equal to ***Calculate***, then the stopping criteria is when the width of the confidence interval for all MOEs (time integral ***TargetValue***, ***MissionTime***, ***FiringRate***, and ***AveFiringRate)*** is less than ***StoppingPercentage*** times their average value. The variables ***StopMode*** and ***StopingPercentage*** are set in the file ***SimParm.Dat.***

C. THE SIMULATION SCENARIO

The base line test scenario used to test the simulation is a hypothetical situation of the bombardment of Iraqi defensive shore installations in the Persian Gulf. The ships will consist of two Spruance class destroyers, each with two 5 inch 54 caliber guns and two Perry class frigates, each with one 76mm OTO Melara gun. The ships are constrained to maneuver on a 5000 yard battle line. The targets will be one headquarters building, two anti-aircraft batteries, two artillery batteries, three infantry squads, and four tanks. This is the base line test scenario and is shown in Figure 3.

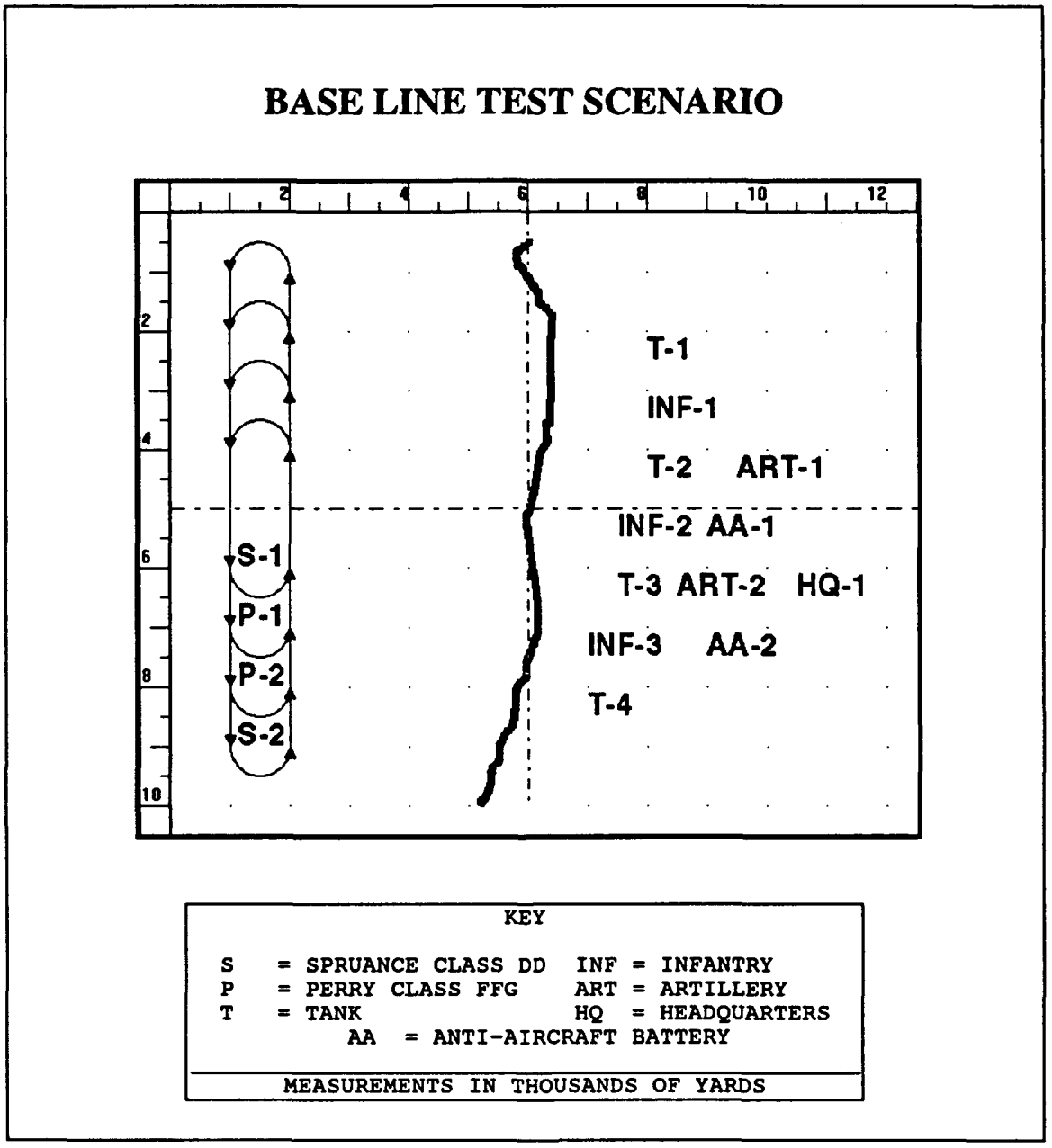

Figure 3. Base Line Test Scenario

D. THE BASE LINE SIMULATION RESULTS

The results of running the simulation on the base line test scenario are presented in Figure 4. It must be cautioned that these results are not absolute only relative. The

numbers themselves have no direct meaning as the model has not been validated or calibrated against actual results. The only value of these numbers is in comparison. Comparing the results of two runs with different inputs will give an idea of the relative effectiveness of the two runs. The results of the test scenario are presented here as an example and a reference point.

The results are presented in an output file that first consists of the date and time stamp indicating when the simulation is started. Next follows a number of repetition records. There is one repetition record for each repetition that is required to be performed to meet the termination conditions of the simulation. In this case the termination condition requires that the confidence interval width for the average value of each MOE be less than 20% of the value of the MOE. This required that 419 repetitions of the scenario be run. Each repetition record contains the firing rate for each gun, the global average firing rate, the mission time, and the time integral target value for that repetition. After the required repetition records are recorded, the end of simulation stats are output. They consist of the total number of repetitions required to meet the termination conditions and the average values of all the MOEs, together with their confidence intervals. Then an execution completion statement and end of simulation date and time stamp completes the output file.

```
********************************************************************************
Output from NGFS at Mon Sep 02 01:44:46 1991
********************************************************************************
*****************************************
 Rep number:     1
      3.94  Firing Rate For DD____-01GUN-1
      2.31  Firing Rate For DD____-01GUN-2
      2.51  Firing Rate For DD____-02GUN-1
      1.61  Firing Rate For DD____-02GUN-2
      4.55  Firing Rate For FF____-01GUN-1
      3.70  Firing Rate For FF____-02GUN-1

      3.10  AveFiringRate
     27.27  MissionTime
   4666.39  IntegralTargetValue
*****************************************
                   .
                   .
                   .
         rep records 2 - 418
                   .
                   .
                   .
*****************************************
 Rep number:   419
      2.76  Firing Rate For DD____-01GUN-1
      2.25  Firing Rate For DD____-01GUN-2
      1.70  Firing Rate For DD____-02GUN-1
      1.92  Firing Rate For DD____-02GUN-2
      1.67  Firing Rate For FF____-01GUN-1
      1.96  Firing Rate For FF____-02GUN-1

      2.04  AveFiringRate
     29.31  MissionTime
   6070.26  IntegralTargetValue
********************************************************************************
********************************************************************************
End of Simulation Stats
       419                  Total number of Repetitions
      4.04 +/-       0.40  Average Firing Rate For DD____-01GUN-1
      2.53 +/-       0.15  Average Firing Rate For DD____-01GUN-2
      3.47 +/-       0.35  Average Firing Rate For DD____-02GUN-1
      2.58 +/-       0.19  Average Firing Rate For DD____-02GUN-2
      4.37 +/-       0.33  Average Firing Rate For FF____-01GUN-1
      4.82 +/-       0.39  Average Firing Rate For FF____-02GUN-1

      3.64 +/-       0.13  Global Average Firing Rate
     24.30 +/-       1.00  Average Mission Time
   4877.09 +/-     187.02  Average Time Integral Target Value
********************************************************************************
********************************************************************************
NGFS Completed execution successfully at Mon Sep 02 06:45:32 1991
********************************************************************************
********************************************************************************
```

Figure 4. Base Line Test Scenario Results

E. ANALYSIS OF THE SIMULATION RESULTS

As this will be part of a larger model, little analysis of actual simulation results has been conducted. That will occur later when the data base and control programs for the eF system have been completed. Several interesting simulation results were noted. One is the fact that for the test scenario, if no failures are allowed to occur in any component, that the average time integral target value was reduced by approximately 43%. This is a very significant improvement and means that reliability plays a large part in the performance of the NGFS system.

Another interesting result is the importance that communications delays play in the results of the simulation. They are the parameter with the most effect on the values of the MOEs. This can be readily seen when the firing rates of the guns are compared to the time allowed for communications between the spotter and the ship. The final result of note is the role that the variable ***TooLong*** plays in the scenario. If a failure occurs and the repair time is longer than ***TooLong***, then the target must be reassigned to a different gun. This means that all the time invested in prosecuting the target to that point is lost and the new gun must start all over again by spotting the target. This results in a considerable penalty in the value of the MOEs. As a result, there should exist an optimum value for ***TooLong*** that will result in optimizing the MOEs for a given scenario.

VI. PROBLEMS ENCOUNTERED

Several types of problems were encountered in the design and implementation of this simulation that have a direct relationship to object-orientation in general and MODSIM II in particular. Several of the major problems and their solutions are described below.

A. SWITCHING TO OBJECT-ORIENTED PROGRAMMING

The redevelopment of a model written in the linear programming language, FORTRAN, into the object-oriented language, MODSIM II proved to be more difficult than expected. The first code that was converted to MODSIM II code tended to be linear code in an object-oriented language and was a patchwork of exceptions and fixes at best. Attempts to fix this were unsuccessful. The final solution was to treat the original attempt as a prototype and once it was working to some degree, set it aside and start over. This allowed the lessons learned in the areas of what objects need to exist, what data they need to contain, what methods they need to do, to be applied to a clean slate. This resulted in much cleaner, more truly object-oriented code, and is well worth the effort in the long run.

B. TIMING AND STATE TRANSITION

Another area that was particularly troublesome was the area of timing and state transition. Timing concerns the order in which events occur in simulation time and in real time. State transition concerns the way the status of an object is changed. Due to the nature of OOP and process-based simulation, there is no linear flow path through the

code and as Fishman stated there is resultant loss of programming control [Ref. 7:pp. 138-139]. This loss of programming control causes problems in timing and state transition. This is because it is difficult to ensure that events that always need to occur in a particular order (timing) are actually executed in that particular order. The loss of programming control also makes it difficult to ensure that the fields of an object are changed correctly (state transition). An example of where the timing problem could occur is in the assignment, by the headquarters object, of a target to a ship and the subsequent assignment, by the ship, of the target to a particular gun. It is obvious that the these events must happen in the order stated, but ensuring that they did was difficult. An example of the state transition problem would be in maintaining the correct status of a gun while it is firing, breaking down, being repaired, and the ship is maneuvering.

The solution to this type of problem is that most of the design work of the model should be put into timing and state transition. The simulation should be kept simple at first, just concentrating on timing and state transition. When the simulation runs correctly and the timing and state transition problems have been solved, then the simulation should be enhanced or expanded to meet the requirements.

One aside on this issue is that problems of this nature mimic the actual problems of communications (timing) and status (state transition) that exist in real life. Consequently, looking at how the problem is handled in real life frequently leads to ideas of how to handle the problem in a program.

C. MULTIPLE CONCURRENT PROCESSES

The multiple concurrent processes that can occur in the simulation language MODSIM II are both a feature and a source of problems. They are a feature in that they can allow one object to be performing many different activities in the form of concurrent methods at the same time. This greatly simplifies the code that is needed in situations

where this concurrence is applicable. The problems arise in keeping track of what an object is doing at any one instance in time. Where the code is specifically designed with concurrence in mind this is frequently not a problem, because the need to keep track of what is going on is taken into account in the design. Where problems really show up, however, is when methods that were not meant to be concurrent happen concurrently. For example, in this simulation, MODSIM II will allow the same ship to maneuver in two different directions at the same time. Obviously this should not occur, but the program will usually execute without error when it does occur with the result that the state of the object in question, in this case the location of the ship, bouncing around between what one method tries to set it as and what the other method tries to set it as.

The solution to this type of problem is to keep track of the processes that are occurring in an object at any given time. This is done is by setting flags and state descriptors. In this simulation this was accomplished with the use of the ***Process*** field in the ship and gun objects. It was not found to be necessary in the design of the other objects.

D. MODEL SIZE LIMITATIONS

The present model is at the practical limit of the size of a model that can be developed and executed on an IBM PC compatible running DOS. The compilation of the larger modules bumped up against the size of the largest module that can be successfully compiled under DOS. If too many output statements were added or if the trace back option of MODSIM II was used, the program could not be compiled successfully.

There was also problem with the time required for execution. The base line test scenario required approximately 5 hours to run on a 386/25MHZ machine. It executed one repetition of the test scenario in approximately 43 seconds. To expand the scenarios and allow this model to be of practical use, it has to run faster.

The solution to both these problems is to move to a more powerful platform. The portability of MODSIM II is very handy in this regard, because the code can be transferred to a Sun workstation, for example, recompiled and executed with extremely few, if any, changes required. In addition, as long as objects do not get as complicated as the gun object in this simulation, the object can be written and debugged on a PC and transferred to a more powerful machine for inclusion in the complete simulation.

There is a version of MODSIM II which will run on a PC under the OS/2 operating system. This would solve the inability to compile large modules, but the speed of execution would still be inadequate. The time required to execute may be successfully addressed by the newer, more powerful PCs coming on the market, specifically the new 486 machines or a faster 386 machine with a math coprocessor.

VII. EXPANSION OF THE BASIC MODEL

One of the goals of this research was to examine the effects that writing the simulation in an OOP language had on the ease with which the original model could be expanded upon. In this area, object-orientation was especially useful. The model was very easily modified. The modifications could usually be developed and tested outside of the main program and then easily incorporated into the main program. A few of the ways in which the model was expanded are described below.

A. PRIORITY OF TARGET ENGAGEMENTS

The initial version of the model assigned targets on the basis of the order in which they were read in from the data file. It was realized that this was limiting, not so much in the base line case, but if the model was to be expanded. Therefore, it was decided to expand the model to engage the targets based on a target priority value. This was accomplished by using both the reusability and inheritance attributes of OOP.

The targets were initially maintained by the ***HQ*** object in a queue object supplied with MODSIM II. It was realized that the targets could be engaged in a priority schema if the queue was changed to a ranked queue based on a target's priority value. This was done. A ***Priority*** field was added to the ***Target*** object. The ***TargetList*** queue was changed to a ranked queue object (also supplied by MODSIM II) and the ranking method was overridden to rank the ***Target*** objects based on their priority with the highest priority corresponding to a priority value of one. Nothing else changed in the model. Now the highest priority target would always be first in the queue and would always be assigned to ship first.

B. DAMAGE MODEL

The damage model in the FORTRAN program [Ref. 2] was the basis for the original damage model in this simulation. It was known that this would be inadequate for different sizes and types of shell, but it was desired to make the gun work with this simple damage model first and enhance it later. The enhancement to the present damage model described in Chapter II took advantage of the modular and data encapsulation features of OOP.

The changes consisted of only changes to the ***Shell*** and the ***Target*** modules and their associated data files. A ***MaxDamage*** field and ***EDR*** field was added to the ***Shell*** object and appropriately initialized. The ***DFList*** was added to the ***Target*** object and appropriately initialized. The rest of the implementation consisted of changing the code in the ***Target*** method ***ImpactRound*** to reflect the new model. Again, the enhancement was very simple with little effect on the model as a whole.

C. RANDOM NUMBER SEEDS

The final enhancement that will be described is that of enhancements done to the way random variables were used in the simulation. The original simulation had one random number stream that generated all random numbers. It was realized that this generated a certain dependence in the simulation that was evident upon changing one input value to one random variable and having the sequence of random numbers change for all subsequent random variables. To avoid this, it was decided to have a separate random number stream for all classes and instances of random variables. In addition, the ability to specify a certain random number seed for a particular random number generator was desired.

This was implemented by the addition of the ***Seed*** module. This module implements two new types of objects. The first is a ***SeederObj*** object that reads in a list

of seeds from the file ***NGFSSEED.DAT.*** The list of seeds is divided into separate lists for ***Ship, Gun,*** and ***Target*** seeds. In response to a request to ***GetNextSeed*** and being passed a name of a seed list, the ***SeederObj*** returns the next random number seed in that seed string. This module defines a new type of object, ***MRandObj***, that inherits the attributes of the MODSIM II provided ***RandObj*** and adds the ability to get a seed from the ***seeder*** object with the method ***GetSeed.*** The variable ***seeder,*** of type ***SeederObj,*** is also defined in this module to be used as the global variable to allow access to the ***seeder*** from all modules.

This enhancement was developed and debugged separately from the main simulation and its integration consisted of adding code to create the ***seeder***, read the lists of seeds, adding new ***MRandObj*** objects where required and telling them to ***GetSeed.*** Again, a major expansion was easily incorporated into the existing simulation due to the nature of OOP.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. OBSERVATIONS

1. NGFS

NGFS was successfully modeled. The effects of component reliability on NGFS system performance can now be analyzed.

2. OBJECT ORIENTED, PROCESS BASED SIMULATION

Object-oriented programming delivers on its promises in a simulation environment. The simulation code was easier to write, debug, maintain, and enhance in nearly all situations encountered than similar code written in the programming language FORTRAN. The process-based simulation concept simplified the design and maintenance of large simulations.

3. CONVERTING TO OBJECT-ORIENTED PROGRAMMING

Converting a simulation that was written in a linear programming language to one that is written in an OOP language is difficult. However, the results justify the additional effort. The features of OOP will pay off over the lifetime of the model.

4. LARGE SIMULATIONS AND THE PC

Large simulation development and execution on a PC under DOS and MODSIM II is presently not feasible. There are too many restrictions. The size of a module is restricted to around 800 lines of code. The number of type definitions is limited by the design of the PC. The execution of complicated programs takes a long time (approximately 5 hours for the test scenario to run to a 20% CI width on a 386/25Mhz machine). The executable and all dynamic variables must fit into the 640K

available on a PC. All these restrictions have the net effect of limiting model development to approximately the size of this simulation. However, the size of the model may increase significantly as CACI releases new versions of MODSIM II that run on a PC under Windows or OS/2.

One key fact to remember is the portability of MODSIM II code. If the simulation gets too big for a PC, it can be transferred to a more powerful machine and executed there. This ensures that the simulations will not be artificially limited by the architecture of the machine on which they run.

5. MODSIM II

MODSIM II is an excellent simulation language. Its object-oriented, process-based approach to simulation results in code that is efficient to write, maintainable and highly expandable. The major flaw of the language is the existence of bugs in the code generated by the compiler. These bugs, mainly involved in the use of interrupts and memory, are very annoying. CACI Corporation is addressing these bugs and plans to fix them in a future release. Another flaw of the language is its lack of adequate Input/Output facilities. The ability to control the input and output to the screen or a file is very limited. To its credit, CACI Corporation addresses the screen portion of this problem with SIMGRAPHICS II [Ref. 10], which is an excellent addition of animation and presentation graphics to MODSIM II. The problem is that currently, SIMGRAPHICS is available only on workstation or higher class machines or PC's running OS/2.

B. SUGGESTIONS FOR FURTHER RESEARCH

1. The next logical step in the research of this model is to complete the data base and control program that will allow it to be used, with relative ease, by the NWSC for its intended purpose of exploring the relative benefits of component reliability on

overall NGFS system performance. This enhancement is presently under development. Due to the size of the model, the simulation will be transferred to a workstation to allow for its continued growth.

2. It would be useful to validate the model against real world data by setting up a real test with actual ships and targets and then modeling it to see if the firing rate and mission time are accurately predicted by the model. This would also allow the model to be calibrated so that it would accurately predict such factors as mission times and types of failures.

3. Increase the fidelity of the model. Modify the control structures to model real life with more fidelity. For example, have the HQ object maintain a list of which target is assigned to which ship and what the status of each ship and target is. This would result in redundant data and increase the complexity and size of the model but make it more realistic. It would also prevent some of the subtle timing and control problems inherent in OOP simulations that occurred in this simulation because the model did not use the same timing and state transition as found in real life. More fidelity could be added by creating a Spotter Object that would allow for errors in the spotting process. Currently the spotter is assumed to do his job with no error. This would increase both the resolution and fidelity of the model as errors in the spotting process constitute a large source of errors in the NGFS system. There are many other ways in which the fidelity of the model could be improved. Finally, it must be cautioned, that the utility of these improvements is probably not worth the trouble unless the question being addressed is directly effected by the improvement.

C. CONCLUSIONS

Overall this research was a success. The goals were achieved. NGFS was successfully modeled and some of the advantages and disadvantages of conducting

simulations in an OOP, process-based simulation environment were documented. There remains much work to be done to realize the potential of this model, especially in the area of making it more user friendly and conducting the validation and calibration of it. Finally, it is worth repeating, that object-oriented programming delivers on its promises and results in models that are more reusable, more reliable, easier to expand and easier to maintain than non-object-oriented programming languages. With the ever increasing complexity of simulations, object-oriented programming is where the future lies.

APPENDIX A NGFS SIMULATION PROGRAM

```
9/11/91
MAIN MODULE NGFS;


{-------------------------------------------------------------------------

PROGRAM NAME:   NGFS
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/16/91
LAST MODIFIED:  8/12/91

DESCRIPTION:

This is the main module of a Naval Gunfire Support Simulation.

-------------------------------------------------------------------------}

FROM UtilMod IMPORT DateTime, ClockTimeSecs;
FROM SIMCTRL IMPORT RunSimulation;

VAR
   TimeStr   : STRING;
   StartTime : INTEGER;

BEGIN
   DateTime(TimeStr);
   StartTime := ClockTimeSecs();
   OUTPUT("NGFS STARTING AT ", TimeStr);

   RunSimulation;

   DateTime(TimeStr);
   OUTPUT("END OF NGFS --- NORMAL TERMINATION ON ",TimeStr);
   OUTPUT("Run took ", ClockTimeSecs()-StartTime, " seconds.");

END MODULE.
```

```
9/11/91
DEFINITION MODULE SIMCTRL;

{-------------------------------------------------------------------------

MODULE NAME:     SIMCTRL
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    8/12/91
LAST MODIFIED:   8/12/91

DESCRIPTION:

This is the definition module that controls the simulation of aspect of the
Naval Gunfire Support Simulation.

-------------------------------------------------------------------------}

PROCEDURE RunSimulation;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE SIMCTRL;


(*-------------------------------------------------------------------------

MODULE NAME:    SIMCTRL
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   8/12/91
LAST MODIFIED:  9/1/91

DESCRIPTION:

This is the module that controls the simulation of aspect of the Naval
Gunfire Support Simulation.

-------------------------------------------------------------------------*)

FROM GLOBALS IMPORT EventLog, LogEvents;
FROM UtilMod IMPORT DateTime, ClockTimeSecs;
FROM SimMod  IMPORT StartSimulation, ResetSimTime;
FROM StatMod IMPORT RTimedStatObj;
FROM HQ      IMPORT HQObj;
FROM IOMod   IMPORT FileUseType(Output);

PROCEDURE RunSimulation;

VAR
   HQ         : HQObj;
   SimDone    : BOOLEAN;
   Rep        : INTEGER;

BEGIN

   NEW(HQ);
   ASK HQ TO CreateScenario;
   Rep := 0;

   REPEAT

      Rep := Rep + 1;
      OUTPUT("Beginning rep ", Rep);
      IF LogEvents
         ASK EventLog TO WriteString("Beginning rep ");
         ASK EventLog TO WriteLnInt(Rep,5);
      END IF;
      ResetSimTime(0.0);
      ASK HQ TO InitializeScenario;
      ASK HQ TO RegisterAllShips;
      StartSimulation;
      OUTPUT("Rep ",Rep," completed.  Integral Target Value = ",
      GETMONITOR(HQ.TargetValue,RTimedStatObj).Sum);
      IF LogEvents
         ASK EventLog TO WriteString("Rep ");
         ASK EventLog TO WriteInt(Rep,5);
         ASK EventLog TO WriteString(" completed.  Integral Target Value = ");
         ASK EventLog TO WriteLnReal(
             GETMONITOR(HQ.TargetValue,RTimedStatObj).Sum,8,2);
```

```
      END IF;
      ASK HQ TO CalculateRunStats(Rep, SimDone);
   UNTIL SimDone;
   ASK HQ TO ReportStats(Rep);
   DISPOSE(HQ);
END PROCEDURE;
END MODULE.
```

```
9/11/91
DEFINITION MODULE HQ;

{-------------------------------------------------------------------------

MODULE NAME:     HQ
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/17/91
LAST MODIFIED:   8/2/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the HQ or
headquarters object.  HQ is responsible for creating the scenario and
assigning targets to ships to be engaged.

-------------------------------------------------------------------------}

FROM StatMod IMPORT SREAL, SINTEGER, TSREAL;
FROM MGrpMod IMPORT ListObj;
FROM GrpMod IMPORT RankedObj;

TYPE

StopModeTYPE = (NumReps, Calculated);

{-------------------------------------------------------------------------}

TargetListObj = OBJECT(RankedObj)
   OVERRIDE
      ASK METHOD Rank(IN a, b : ANYOBJ) : INTEGER;
END OBJECT;

{-------------------------------------------------------------------------}

HQObj = OBJECT

   StopSim            : BOOLEAN;
   MissionTime        : SREAL;
   AveFiringRate      : SREAL;
   TargetValue        : TSREAL;
   IntegralTargetValue: SREAL;
   StopMode           : StopModeTYPE;
   MaxRep             : INTEGER;
   StoppingPercentage : REAL;
   InvNormalCI        : REAL;
   ShipList           : ListObj;
   TargetList         : TargetListObj;

   ASK METHOD CreateScenario;
   ASK METHOD InitializeScenario;
   ASK METHOD CalculateRunStats(IN  Rep     : INTEGER;
                                OUT SimDone : BOOLEAN);
   ASK METHOD ReportStats(IN Rep : INTEGER);
   ASK METHOD UpdateTargetValue;
   ASK METHOD RegisterAllShips;
```

```
   TELL METHOD AssignTargets;

END OBJECT;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE HQ;
(*-------------------------------------------------------------------------

MODULE NAME:    HQ
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/17/91
LAST MODIFIED:  9/1/91

DESCRIPTION:

This is a module of the NGFS simulation that implements the HQ or
headquarters object.  HQ is responsible for creating the scenario and
assigning targets to ships to be engaged.

-------------------------------------------------------------------------*)

FROM GLOBALS IMPORT ShellList, xyPoint, TooLong, LOG, SpotDeltaT,
                    SpecifiedShellRecord, LogEvents, EventLog;
FROM TARGET  IMPORT TargetObj, ALL TargetStatusTYPE;
FROM SHIP    IMPORT ShipObj,   ALL EngagementStatusTYPE;
FROM GUN     IMPORT GunObj;
FROM MIOMod  IMPORT MStreamObj, InputLog;
FROM IOMod   IMPORT FileUseType(Input, Output);
FROM SHELL   IMPORT ShellObj;
FROM SimMod  IMPORT SimTime, PendingListDump, Interrupt,InterruptAll,
                    ActivityListDump;
FROM Debug   IMPORT TraceStream;
FROM StatMod IMPORT RTimedStatObj, RStatObj;
FROM MathMod IMPORT SQRT;
FROM UtilMod IMPORT DateTime;
FROM SEED    IMPORT seeder;

(*-------------------------------------------------------------------------*)
OBJECT TargetListObj;
   ASK METHOD Rank(IN a, b : ANYOBJ) : INTEGER;

   VAR
      Target1 : TargetObj;
      Target2 : TargetObj;

   BEGIN
      Target1 := a;
      Target2 := b;
      IF Target1.Priority > Target2.Priority
         RETURN 1;
      ELSIF Target1.Priority = Target2.Priority
         RETURN 0;
      ELSE
         RETURN -1;
      END IF;
   END METHOD;
END OBJECT;

(*-------------------------------------------------------------------------*)
```

```
OBJECT HQObj;

ASK METHOD CreateScenario;

VAR
   TimeStr          : STRING;
   SceneFile        : MStreamObj;
   SimFile          : MStreamObj;
   StopModeString   : STRING;
   NumShips         : INTEGER;
   NumTargets       : INTEGER;
   NumShells        : INTEGER;
   Ship             : ShipObj;
   Target           : TargetObj;
   Shell            : ShellObj;
   I                : INTEGER;
   Type             : STRING;
   ID               : STRING;
   Position         : xyPoint;
   ShipCourse       : REAL;
   ShipSpeed        : REAL;
   TargetPriority   : INTEGER;
   TargetFOV        : INTEGER;
   ManFileName      : STRING;
   SpecifiedShells  : SpecifiedShellRecord;
   ThisShell        : SpecifiedShellRecord;
   PreviousSpecifiedShell : SpecifiedShellRecord;
   SeedFile         : STRING;
   LogEventsStr      : STRING;
   tempstr          : STRING;


BEGIN
   NEW(InputLog);
   ASK InputLog TO Open("INPUT.LOG", Output);
   NEW(LOG);
   ASK LOG TO Open("NGFS.OUT",Output);
   ASK LOG TO WriteLnStr(
"*******************************************************************************");
   ASK LOG TO WriteString("Output from NGFS at ");
   DateTime(TimeStr);
   ASK LOG TO WriteLnStr(TimeStr);
   ASK LOG TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteString("Input Log File for NGFS run at ");
   ASK InputLog TO WriteLnStr(TimeStr);
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   NEW(SimFile);
   ASK SimFile TO Open("SimParm.dat", Input);
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr("Simulation Parameters");
   ASK SimFile TO SkipLines(12);
   ASK SimFile TO ReadLnStrLOG(StopModeString,"StopMode");
```

```
   IF SUBSTR(1,1,StopModeString) = "C"
      StopMode := Calculated;
   ELSE
      StopMode := NumReps;
   END IF;
   ASK SimFile TO ReadLnIntLOG(MaxRep,"MaxRep");
   ASK SimFile TO ReadLnRealLOG(StoppingPercentage,"StoppingPercentage");
   ASK SimFile TO SkipLines(1);
   ASK SimFile TO ReadLnRealLOG(InvNormalCI,"InvNormalCI");
   ASK SimFile TO ReadLnStrLOG(SeedFile,"SeedFile name");
   ASK SimFile TO ReadLnStrLOG(LogEventsStr,"LogEvents");
   IF SUBSTR(1,1,LogEventsStr) = "T"
      LogEvents := TRUE;
   ELSE
      LogEvents := FALSE;
   END IF;
   NEW(seeder);
   ASK seeder TO ReadSeeds(SeedFile);
   ASK SimFile TO Close;
   DISPOSE(SimFile);

   NEW(ShellList);

   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr("Scenario Data");
   NEW (SceneFile);
   ASK SceneFile TO Open("ACTUAL.SCN", Input);
   ASK SceneFile TO SkipLines(11);
   ASK SceneFile TO ReadLnRealLOG(TooLong,"TooLong");
   ASK SceneFile TO SkipLines(3);

   NEW(TargetList);
   ASK SceneFile TO ReadLnIntLOG(NumTargets,"NumTargets");
   ASK InputLog TO WriteLn;
   ASK SceneFile TO SkipLines(3);
   FOR I:=1 TO NumTargets
      ASK InputLog TO WriteLnStr(
"TargetType    TargetID      Xpos     Ypos   Priority  FOV  Specified Shells");
      ASK InputLog TO WriteLnStr(
"*******************************************************************************");
      ASK SceneFile TO ReadString(Type);
      tempstr := "           ";
      REPLACE(tempstr,13-STRLEN(Type),12,Type);
      ASK InputLog TO WriteString(tempstr);
      ASK SceneFile TO ReadString(ID);
      tempstr := "           ";
      REPLACE(tempstr,12-STRLEN(ID),11,ID);
      ASK InputLog TO WriteString(tempstr);
      ASK SceneFile TO ReadReal(Position.x);
      ASK InputLog TO WriteReal(Position.x,10,2);
      ASK SceneFile TO ReadReal(Position.y);
      ASK InputLog TO WriteReal(Position.y,9,2);
      ASK SceneFile TO ReadInt(TargetPriority);
      ASK InputLog TO WriteInt(TargetPriority,6);
      ASK SceneFile TO ReadInt(TargetFOV);
      ASK InputLog TO WriteInt(TargetFOV,7);
      ASK InputLog TO WriteString("   ");
```

```
      NEW(SpecifiedShells);
      ThisShell := SpecifiedShells;
      ASK SceneFile TO ReadString(ThisShell.ShellKind);
      ASK InputLog TO WriteString(ThisShell.ShellKind);
      IF ThisShell.ShellKind <> "none"
         REPEAT
            PreviousSpecifiedShell := ThisShell;
            NEW(ThisShell);
            PrevicusSpecifiedShell.NextShell := ThisShell;
            ASK SceneFile TO ReadString(ThisShell.ShellKind);
            ASK InputLog TO WriteString(" "+ThisShell.ShellKind);
         UNTIL ThisShell.ShellKind = "x";
         ASK InputLog TO WriteLn;
         ASK InputLog TO WriteLnStr(
**************************************************************************");
         DISPOSE(ThisShell);
         PreviousSpecifiedShell.NextShell := NILREC;
      ELSE
         ASK InputLog TO WriteLn;
         ASK InputLog TO WriteLnStr(
**************************************************************************");
         DISPOSE(SpecifiedShells);
         SpecifiedShells := NILREC;
      END IF;

      NEW(Target);
      ASK Target TO CreateTarget(Type, ID, Position, TargetPriority,
                                 TargetFOV, SpecifiedShells, SELF);
      ASK TargetList TO Add(Target);
   END FOR;

{
Target := ASK TargetList First();
WHILE Target <> NILOBJ
OUTPUT(Target.ID," priority = ",Target.Priority);
Target := ASK TargetList Next(Target);
END WHILE;
}
   ASK SceneFile TO SkipLines(3);

   ASK InputLog TO WriteLnStr(
**************************************************************************");
   ASK SceneFile TO ReadLnIntLOG(NumShips,"NumShips");
   ASK InputLog TO WriteLn;
   NEW(ShipList);
   ASK SceneFile TO SkipLines(3);
   FOR I:=1 TO NumShips
   ASK InputLog TO WriteLnStr(
"ShipType       ShipID      Xpos      Ypos   Course   Speed  ManFileName");
      ASK InputLog TO WriteLnStr(
**************************************************************************");
      ASK SceneFile TO ReadString(Type);
      tempstr := "            ";
      REPLACE(tempstr,13 - STRLEN(Type),12,Type);
      ASK InputLog TO WriteString(tempstr);
      ASK SceneFile TO ReadString(ID);
      tempstr := "            ";
      REPLACE(tempstr,12 - STRLEN(ID),11,ID);
```

```
      ASK InputLog TO WriteString(tempstr);
      ASK SceneFile TO ReadReal(Position.x);
      ASK InputLog TO WriteReal(Position.x,10,1);
      ASK SceneFile TO ReadReal(Position.y);
      ASK InputLog TO WriteReal(Position.y,9,1);
      ASK SceneFile TO ReadReal(ShipCourse);
      ASK InputLog TO WriteReal(ShipCourse,8,1);
      ASK SceneFile TO ReadReal(ShipSpeed);
      ASK InputLog TO WriteReal(ShipSpeed,7,1);
      ASK SceneFile TO ReadLnStr(ManFileName);
      ASK InputLog TO WriteLnStr("    "+ManFileName);
      ASK InputLog TO WriteLnStr(
"*******************************************************************************");

      NEW(Ship);
      ASK Ship TO CreateShip(Type, ID, Position, ShipCourse,
                             ShipSpeed, ManFileName, SELF);
      ASK ShipList TO Add(Ship);
   END FOR;

   ASK SceneFile TO Close;
   DISPOSE(SceneFile);
   ASK InputLog TO WriteLn;
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr("End of NGFS Input Log");
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");
   ASK InputLog TO WriteLnStr(
"*******************************************************************************");

   ASK InputLog TO Close;
   DISPOSE(InputLog);
   DISPOSE(seeder);
   IF LogEvents
      NEW(EventLog);
      ASK EventLog TO Open("Event.LOG", Output);
      ASK EventLog TO WriteLnStr(
"*******************************************************************************");
      ASK EventLog TO WriteString("EventLog from NGFS at ");
      ASK EventLog TO WriteLnStr(TimeStr);
      ASK EventLog TO WriteLnStr(
"*******************************************************************************");

   END IF;


OUTPUT("CREATED SCENARIO");

END METHOD;


{-------------------------------------------------------------------------------}
```

```
ASK METHOD InitializeScenario;

VAR

   Ship   : ShipObj;
   Target : TargetObj;

BEGIN

   Ship := ASK ShipList First();
   WHILE Ship <> NILOBJ
      ASK Ship TO InitializeShip;
      Ship := ASK ShipList Next(Ship);
   END WHILE;

   Target := ASK TargetList First();
   WHILE Target <> NILOBJ
      ASK Target TO InitializeTarget;
      Target := ASK TargetList Next(Target);
   END WHILE;

   StopSim := FALSE;
   ASK GETMONITOR(TargetValue, RTimedStatObj) TO Reset;
   ASK SELF TO UpdateTargetValue;

END METHOD;

{-------------------------------------------------------------------------}
ASK METHOD UpdateTargetValue;

VAR
   Subtotal : REAL;
   Target   : TargetObj;

BEGIN
   Subtotal := 0.0;
   Target := ASK TargetList First();
   WHILE Target <> NILOBJ
      Subtotal := Subtotal + Target.LifePoints;
      Target := ASK TargetList Next(Target);
   END WHILE;
   TargetValue := Subtotal;
   IF LogEvents
      ASK EventLog TO WriteString("SimTime = ");
      ASK EventLog TO WriteReal(SimTime(),8,2);
      ASK EventLog TO WriteString(" Current Target Value = ");
      ASK EventLog TO WriteLnReal(TargetValue,8,2);
      IF SimTime() > 0.0
         ASK EventLog TO WriteString("Current Time Integral Value = ");
         ASK EventLog TO WriteLnReal(GETMONITOR(TargetValue,
            RTimedStatObj).Sum,8,2);
      END IF;
   END IF;

END METHOD;


{-------------------------------------------------------------------------}
```

```
ASK METHOD CalculateRunStats(IN  Rep     : INTEGER;
                             OUT SimDone : BOOLEAN);

VAR
   I       : INTEGER;
   RPM     : REAL;
   SumRPM  : REAL;
   GunDone : BOOLEAN;
   Ship    : ShipObj;
   Gun     : GunObj;

BEGIN

   SimDone := TRUE;
   ASK LOG TO WriteLnStr("*****************************************");
   ASK LOG TO WriteString(" Rep number: ");
   ASK LOG TO WriteLnInt(Rep,5);
   Ship := ASK ShipList First();
   WHILE Ship <> NILOBJ
      Gun := ASK Ship.GunList First();
      WHILE Gun <> NILOBJ
         I := I + 1;
         ASK Gun TO CalRunStats(Rep, StoppingPercentage, InvNormalCI,
                                RPM, GunDone);
         SumRPM := SumRPM + RPM;
         SimDone := SimDone AND GunDone;
         ASK LOG TO WriteReal(RPM,10,2);
         ASK LOG TO WriteString("  Firing Rate For ");
         ASK LOG TO WriteLnStr(Gun.ID);
         Gun := ASK Ship.GunList Next(Gun);
      END WHILE;
      Ship := ASK ShipList Next(Ship);
   END WHILE;
   AveFiringRate := SumRPM/FLOAT(I);
   IntegralTargetValue := GETMONITOR(TargetValue, RTimedStatObj).Sum;
   ASK LOG TO WriteLn;
   ASK LOG TO WriteReal(AveFiringRate,10,2);
   ASK LOG TO WriteLnStr("  AveFiringRate  ");
   ASK LOG TO WriteReal(MissionTime,10,2);
   ASK LOG TO WriteLnStr("  MissionTime    ");
   ASK LOG TO WriteReal(IntegralTargetValue,10,2);
   ASK LOG TO WriteLnStr("  IntegralTargetValue ");
   IF StopMode = NumReps
      SimDone := (Rep >= MaxRep);
   ELSIF Rep >= 10
      SimDone := SimDone AND ((StoppingPercentage*
                   ASK GETMONITOR(AveFiringRate,RStatObj) Mean()) >=
                   (2.0*InvNormalCI*
                   (ASK GETMONITOR(AveFiringRate,RStatObj) StdDev()/
                   SQRT(FLOAT(Rep)-1.0))));
      SimDone := SimDone AND ((StoppingPercentage*
                   ASK GETMONITOR(IntegralTargetValue,RStatObj) Mean()) >=
                   (2.0*InvNormalCI*
                   (ASK GETMONITOR(IntegralTargetValue,RStatObj) StdDev()/
                   SQRT(FLOAT(Rep)-1.0))));
      SimDone := SimDone AND ((StoppingPercentage*
                   ASK GETMONITOR(MissionTime,RStatObj) Mean()) >=
```

```
                    (2.0*InvNormalCI*
                    (ASK GETMONITOR(MissionTime,RStatObj) StdDev()/
                    SQRT(FLOAT(Rep)-1.0))));
    ELSE
       SimDone := FALSE;
    END IF;

END METHOD;

{-------------------------------------------------------------------------}
ASK METHOD ReportStats(IN Rep : INTEGER);

VAR
   Gun     : GunObj;
   Ship    : ShipObj;
   TimeStr : STRING;

BEGIN

   ASK LOG TO WriteLnStr(
"*************************************************************************");
   ASK LOG TO WriteLnStr(
"*************************************************************************");
   ASK LOG TO WriteLnStr("End of Simulation Stats");
   ASK LOG TO WriteInt(Rep,10);
   ASK LOG TO WriteLnStr("                Total number of Repetitions");
   Ship := ASK ShipList First();
   WHILE Ship <> NILOBJ
      Gun := ASK Ship.GunList First();
      WHILE Gun <> NILOBJ
         ASK LOG TO WriteReal(ASK GETMONITOR(Gun.AveFiringRate,RStatObj)
            Mean(),10,2);
         ASK LOG TO WriteString(" +/-");
         ASK LOG TO WriteReal(InvNormalCI*
                    (ASK GETMONITOR(Gun.AveFiringRate,RStatObj) StdDev()/
                    SQRT(FLOAT(Rep)-1.0)),10,2);
         ASK LOG TO WriteString("  Average Firing Rate For ");
         ASK LOG TO WriteLnStr(Gun.ID);
         Gun := ASK Ship.GunList Next(Gun);
      END WHILE;
      Ship := ASK ShipList Next(Ship);
   END WHILE;
   ASK LOG TO WriteLn;

   ASK LOG TO WriteReal(ASK GETMONITOR(AveFiringRate,RStatObj) Mean(),10,2);
   ASK LOG TO WriteString(" +/-");
   ASK LOG TO WriteReal(InvNormalCI*
                    (ASK GETMONITOR(AveFiringRate,RStatObj) StdDev()/
                    SQRT(FLOAT(Rep)-1.0)),10,2);
   ASK LOG TO WriteLnStr("  Global Average Firing Rate");
   ASK LOG TO WriteReal(ASK GETMONITOR(MissionTime,RStatObj) Mean(),10,2);
   ASK LOG TO WriteString(" +/-");
   ASK LOG TO WriteReal(InvNormalCI*
                    (ASK GETMONITOR(MissionTime,RStatObj) StdDev()/
                    SQRT(FLOAT(Rep)-1.0)),10,2);
   ASK LOG TO WriteLnStr("  Average Mission Time");
   ASK LOG TO WriteReal(ASK GETMONITOR(IntegralTargetValue,RStatObj) Mean(),
                        10,2);
```

```
   ASK LOG TO WriteString(" +/-");
   ASK LOG TO WriteReal(InvNormalCI*
                   (ASK GETMONITOR(IntegralTargetValue,RStatObj) StdDev()/
                   SQRT(FLOAT(Rep)-1.0)),10,2);
   ASK LOG TO WriteLnStr("  Average Time Integral Target Value");
   ASK LOG TO WriteLnStr(
*****************************************************************************');
   ASK LOG TO WriteLnStr(
*****************************************************************************");
   ASK LOG TO WriteString("NGFS Completed execution successfully at ");
   DateTime(TimeStr);
   ASK LOG TO WriteLnStr(TimeStr);
   ASK LOG TO WriteLnStr(
*****************************************************************************");
   ASK LOG TO WriteLnStr(
*****************************************************************************");

   ASK LOG TO Close;
   DISPOSE(LOG);
   IF LogEvents
   ASK EventLog TO WriteLnStr(
*****************************************************************************");
   ASK EventLog TO WriteLnStr(
*****************************************************************************");
   ASK EventLog TO WriteString("NGFS Completed execution successfully at ");
   ASK EventLog TO WriteLnStr(TimeStr);
   ASK EventLog TO WriteLnStr(
*****************************************************************************");
   ASK EventLog TO WriteLnStr(
*****************************************************************************");
      ASK EventLog TO Close;
      DISPOSE(EventLog);
   END IF;

END METHOD;

{-----------------------------------------------------------------------------}

ASK METHOD RegisterAllShips;

VAR
   Ship : ShipObj;

BEGIN
   Ship := ASK ShipList First();
   WHILE Ship <> NILOBJ
      ASK Ship TO RegistrationFire;
      Ship := ASK ShipList Next(Ship);
   END WHILE;
END METHOD;

{-----------------------------------------------------------------------------}

TELL METHOD AssignTargets;

VAR
   Target       : TargetObj;
   Assigned     : BOOLEAN;
```

```
   Ship         : ShipObj;
   AllDestroyed : BOOLEAN;
BEGIN

   IF LogEvents
      ASK EventLog TO WriteString("HELLO FROM ASSIGN TARGETS SIMTIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

   Target := ASK TargetList First();
   Assigned := TRUE;
   AllDestroyed := TRUE;
   REPEAT
      AllDestroyed := AllDestroyed AND (Target.Status = Destroyed);
      IF (Target.Status <> Destroyed) AND (NOT Target.Assigned)
         Assigned := FALSE;
         Ship := ASK ShipList First();
         WHILE (Ship <> NILOBJ) AND (NOT Assigned)
            IF (((ASK Ship EngagementStatus) < FullyEngaged) AND
                                    (ASK Ship RegistrationComplete))
               IF LogEvents
                  ASK EventLog TO WriteLnStr(Target.ID+" assigned to
"+Ship.ID);
               END IF;
               ASK Ship TO EngageTarget(Target);
               Assigned := TRUE;
            ELSE
               Ship := ASK ShipList Next(Ship);
            END IF;
         END WHILE;
      END IF;
      Target := ASK TargetList Next(Target);
   UNTIL (Target = NILOBJ) OR NOT Assigned;

   IF AllDestroyed AND NOT StopSim
      StopSim := TRUE;
      MissionTime := SimTime();
      Ship := ASK ShipList First();
      WHILE Ship <> NILOBJ
         ASK Ship TO HaltManeuvers;
         Ship := ASK ShipList Next(Ship);
      END WHILE;

      IF LogEvents
         ASK EventLog TO WriteLnStr(
**************    ALL TARGETS DESTROYED -- TERMINATING  ******************");
         ASK EventLog TO WriteString("                          SIMTIME = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
   END IF;

   IF LogEvents
      IF Assigned
         ASK EventLog TO WriteLnStr("LEAVING ASSIGN TARGETS Assigned = TRUE");
      ELSE
         ASK EventLog TO WriteLnStr("LEAVING ASSIGN TARGETS Assigned = FALSE");
      END IF;
   END IF;
```

```
END METHOD;

END OBJECT;

END MODULE.
```

```
9/11/91
DEFINITION MODULE SHIP;


(-------------------------------------------------------------------------

MODULE NAME:     SHIP
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/23/91
LAST MODIFIED:   8/1/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the Ship object.
Ship is responsible for creating itself and assigning targets to its
guns to be engaged.  It also manuevers and keeps track of its location.


-------------------------------------------------------------------------}
FROM MGrpMod IMPORT IDObj, ListObj;
FROM TARGET  IMPORT TargetObj, ALL TargetStatusTYPE;
FROM GLOBALS IMPORT xyPoint;
FROM SEED    IMPORT MRandomObj;
FROM SimMod  IMPORT TriggerObj;
FROM GUN     IMPORT GunObj;


TYPE

ShipStatusTYPE = (OK, Degraded);
EngagementStatusTYPE = ( UnEngaged, Engaged, FullyEngaged, Down);
ShipProcessTYPE = (pIdle, pRegistration, pAwaitingRegistration);

ManeuverStatusTYPE = (NoneScheduled, AwaitingManeuver, WaitingForCTMt,
                      Maneuvering);
ManeuverRecord = RECORD
   DTime          : REAL;
   DCourse        : REAL;
   DSpeed         : REAL;
   Advance        : REAL;
   Transfer       : REAL;
   Duration       : REAL;
   NextManeuver   : ManeuverRecord;
END RECORD;

ShipObj = OBJECT(IDObj);

   Type                 : STRING;
   Status               : ShipStatusTYPE;
   EngagementStatus     : EngagementStatusTYPE;
   Process              : ShipProcessTYPE;
   StopManeuvering      : BOOLEAN;
   ManeuverStatus       : ManeuverStatusTYPE;
   LastLocation         : xyPoint;
   TimeLastLocation     : REAL;
   Course               : REAL;
   Speed                : REAL;
   NavError             : xyPoint;
```

```
   NavSigma             : xyPoint;
   Bias                 : xyPoint;
   HQPtr                : ANYOBJ;
   NumberOfGuns         : INTEGER;
   GunList              : ListObj;
   RegistrationComplete : BOOLEAN;
   NextManeuver         : ManeuverRecord;
   ClearedToManeuver    : TriggerObj;

   ASK METHOD CreateShip(IN InShipType  : STRING;
                         IN ShipID      : STRING;
                         IN InLocation  : xyPoint;
                         IN InCourse    : REAL;
                         IN InSpeed     : REAL;
                         IN ManFileName : STRING;
                         IN InHQ        : ANYOBJ);

   ASK METHOD InitializeShip;
   ASK METHOD UpdateEngagementStatus(IN ReAssignTargets : BOOLEAN);
   ASK METHOD UpdatePosition;
   ASK METHOD ReceiveRegStatus(IN Succesful : BOOLEAN;
                               IN InBias    : xyPoint;
                               IN OldGun    : GunObj);
   ASK METHOD RecManvGranted;
   ASK METHOD HaltManeuvers;
   ASK METHOD DropAll;
   ASK METHOD EngageTarget(IN Target : TargetObj);
   ASK METHOD RegistrationFire;

   TELL METHOD Maneuver(IN ManvOrder : ManeuverRecord);

PRIVATE
   rLocation     : xyPoint;
   rCourse       : REAL;
   rSpeed        : REAL;
   rNextManeuver : ManeuverRecord;
   RandomGen     : MRandomObj;

END OBJECT;
END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE SHIP;


(*-------------------------------------------------------------------------

MODULE NAME:    SHIP
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/23/91
LAST MODIFIED:  9/2/91

DESCRIPTION:

This is a module of the NGFS simulation that implements the Ship object.
Ship is responsible for creating itself and assigning targets to its
guns to be engaged.  It also manuevers and keeps track of its location.


-------------------------------------------------------------------------*)

FROM MIOMod  IMPORT MStreamObj,InputLog;
FROM IOMod   IMPORT FileUseType(Input);
FROM TARGET  IMPORT TargetObj, ALL TargetStatusTYPE;
FROM HQ      IMPORT HQObj;
FROM GLOBALS IMPORT xyPoint, TooLong, ShellList, origin, LogEvents, EventLog;
FROM GUN     IMPORT GunObj, ALL GunStatusTYPE;
FROM MathMod IMPORT SIN, COS, pi;
FROM SimMod  IMPORT SimTime, Interrupt;

OBJECT ShipObj;

ASK METHOD CreateShip(IN InShipType  : STRING;
                      IN ShipID      : STRING;
                      IN InLocation  : xyPoint;
                      IN InCourse    : REAL;
                      IN InSpeed     : REAL;
                      IN ManFileName : STRING;
                      IN InHQ        : ANYOBJ);

VAR
   ShipFile : MStreamObj;
   ManFile  : MStreamObj;
   HQ       : HQObj;
   Gun      : GunObj;
   I        : INTEGER;
   TempManeuver    : ManeuverRecord;
   OldTempManeuver : ManeuverRecord;
   Done     : BOOLEAN;
   GunType  : STRING;
   tempstr  : STRING;

BEGIN

   HQPtr        := InHQ;
   Type         := InShipType;
   ID           := ShipID;
   LastLocation := InLocation;
   Course       := InCourse;
```

```
   Speed    := InSpeed;

   NEW(ShipFile);
   ASK ShipFile TO Open(InShipType+".SHP", Input);
   ASK ShipFile TO SkipLines(9);
   ASK ShipFile TO ReadLnRealLOG(NavSigma.x,"NavSigma.x");
   ASK ShipFile TO ReadLnRealLOG(NavSigma.y,"NavSigma.y");
   NEW(RandomGen);
   ASK RandomGen TO GetSeed("Ship");
   ASK ShipFile TO ReadLnIntLOG(NumberOfGuns,"NumberOfGuns");
   ASK InputLog TO WriteLn;
   ASK ShipFile TO SkipLines(3);
   NEW(GunList);
   FOR I := 1 TO NumberOfGuns
      ASK InputLog TO WriteLnStr(
"*************************************************************");
      ASK ShipFile TO ReadLnStr(GunType);
      tempstr := "            ";
      REPLACE(tempstr,13-STRLEN(GunType),12,GunType);
      ASK InputLog TO WriteLnStr(tempstr);
      ASK InputLog TO WriteLnStr(
"*************************************************************");
      NEW(Gun);
      ASK GunList TO Add(Gun);
      ASK Gun TO CreateGun(GunType,SELF,I);
   END FOR;
   ASK ShipFile TO Close;
   DISPOSE(ShipFile);

   NEW(ManFile);
   ASK ManFile TO Open(ManFileName, Input);
   ASK ManFile TO SkipLines(15);
   ASK InputLog TO WriteLnStr(
"*************************************************************************");
   tempstr :="            ";
   REPLACE(tempstr,13-STRLEN(ManFileName),12,ManFileName);
   ASK InputLog TO WriteLnStr(tempstr+"    Maneuver File");
   ASK InputLog TO WriteLnStr(
"*************************************************************************");
   ASK InputLog TO WriteLn;
   ASK InputLog TO WriteLnStr(
"DeltaTime   DeltaCourse  DeltaSpeed  Advance   Transfer   Duration");
   ASK InputLog TO WriteLn;

   NEW(NextManeuver);
   ASK ManFile TO ReadReal(NextManeuver.DTime);
   ASK InputLog TO WriteReal(NextManeuver.DTime,7,2);
   IF NextManeuver.DTime <= 0.0
      ASK InputLog TO WriteLn;
      DISPOSE(NextManeuver);
      NextManeuver := NILREC;
   ELSE
      TempManeuver := NextManeuver;
      Done := FALSE;

      WHILE NOT Done
         IF TempManeuver.DTime > 0.0
            ASK ManFile TO ReadReal(TempManeuver.DCourse);
```

```
            ASK InputLog TO WriteReal(TempManeuver.DCourse,13,2);
            ASK ManFile TO ReadReal(TempManeuver.DSpeed);
            ASK InputLog TO WriteReal(TempManeuver.DSpeed,12,2);
            ASK ManFile TO ReadReal(TempManeuver.Advance);
            ASK InputLog TO WriteReal(TempManeuver.Advance,12,2);
            ASK ManFile TO ReadReal(TempManeuver.Transfer);
            ASK InputLog TO WriteReal(TempManeuver.Transfer,11,2);
            ASK ManFile TO ReadLnReal(TempManeuver.Duration);
            ASK InputLog TO WriteLnReal(TempManeuver.Duration,9,2);
            OldTempManeuver := TempManeuver;
            NEW(TempManeuver);
            OldTempManeuver.NextManeuver := TempManeuver;
            ASK ManFile TO ReadReal(TempManeuver.DTime);
            ASK InputLog TO WriteReal(TempManeuver.DTime,7,2);
         ELSIF TempManeuver.DTime = 0.0
            OldTempManeuver.NextManeuver := NILREC;
            ASK InputLog TO WriteLn;
            DISPOSE(TempManeuver);
            Done := TRUE;
         ELSE
            OldTempManeuver.NextManeuver := NextManeuver;
            ASK InputLog TO WriteLn;
            DISPOSE(TempManeuver);
            Done := TRUE;
         END IF;
      END WHILE;
   END IF;
   rNextManeuver := NextManeuver;

   ASK InputLog TO WriteLn;
   ASK ManFile TO Close;
   DISPOSE(ManFile);

   rLocation := LastLocation;
   rCourse   := Course;
   rSpeed    := Speed;

   NEW(ClearedToManeuver);


END METHOD;

{-------------------------------------------------------------------------------}


ASK METHOD InitializeShip;

VAR
   Gun : GunObj;

BEGIN

   Process := pIdle;
   LastLocation := rLocation;
   Course   := rCourse;
   Speed    := rSpeed;
   StopManeuvering := FALSE;
   ManeuverStatus := NoneScheduled;
```

```
   NextManeuver := rNextManeuver;
   Status    := OK;
   EngagementStatus := UnEngaged;
   TimeLastLocation := 0.0;
   NavError.x := ASK RandomGen Normal(0.0, NavSigma.x);
   NavError.y := ASK RandomGen Normal(0.0, NavSigma.y);
   Bias := origin;
   RegistrationComplete := FALSE;



   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ;
      ASK Gun TO InitializeGun;
      Gun := ASK GunList Next(Gun);
   END WHILE;

   IF NextManeuver <> NILREC
      TELL SELF TO Maneuver(NextManeuver);
   END IF;

END METHOD;

{------------------------ ------------------------------------------------}


ASK METHOD UpdateEngagementStatus(IN ReAssignTargets : BOOLEAN);

VAR
   Gun : GunObj;
   AllEngagedOrBroke : BOOLEAN;
   AllIdle : BOOLEAN;
   HQ : HQObj;

BEGIN
   HQ := HQPtr;
   AllEngagedOrBroke := TRUE;
   AllIdle :=TRUE;
   IF Process = pAwaitingRegistration
      ASK SELF TO RegistrationFire;
   ELSE
      Gun := ASK GunList First();
      WHILE Gun <> NILOBJ
         AllEngagedOrBroke := AllEngagedOrBroke AND (ASK Gun Status >= Busy);
         AllIdle := AllIdle AND (ASK Gun Status = Idle);
         Gun := ASK GunList Next(Gun);
      END WHILE;
      IF AllEngagedOrBroke
         EngagementStatus := FullyEngaged;
      ELSIF AllIdle
         EngagementStatus := UnEngaged;
      ELSE
         EngagementStatus := Engaged;
      END IF;
      IF ReAssignTargets
         TELL HQ TO AssignTargets;
      END IF;
   END IF;
```

```
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD UpdatePosition;

CONST
   SpeedFac    = 100.0/3.0;
   DegRad      = pi/180.0;

BEGIN
   LastLocation.x := LastLocation.x + SIN(Course*DegRad)*Speed*SpeedFac*
                     (SimTime()-TimeLastLocation);
   LastLocation.y := LastLocation.y + COS(Course*DegRad)*Speed*SpeedFac*
                     (SimTime()-TimeLastLocation);
   TimeLastLocation := SimTime();
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD ReceiveRegStatus(IN Successful : BOOLEAN;
                            IN InBias     : xyPoint;
                            IN OldGun     : GunObj);

VAR
   Gun : GunObj;
   GunFound : BOOLEAN;

BEGIN
   IF LogEvents
      IF Successful
         ASK EventLog TO WriteString(
          "ReceiveRegStatus Reg GOOD, BIAS, oldgunid = TRUE ");
      ELSE
         ASK EventLog TO WriteString(
          "ReceiveRegStatus Reg GOOD, BIAS, oldgunid = FALSE ");
      END IF;
      ASK EventLog TO WriteReal(InBias.x,8,2);
      ASK EventLog TO WriteReal(InBias.y,8,2);
      ASK EventLog TO WriteLnStr(OldGun.ID);
   END IF;
   IF Successful
      RegistrationComplete := TRUE;
      Bias := InBias;
      Process := pIdle;
      ASK SELF TO UpdateEngagementStatus(TRUE);
   ELSE
      ASK SELF TO RegistrationFire;
   END IF;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD RecManyGranted;

VAR
   Gun : GunObj;
   AllGunsClear : BOOLEAN;
```

```
BEGIN

   AllGunsClear := TRUE;
   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ;
      AllGunsClear := AllGunsClear AND Gun.ClearedToManeuver;
      Gun := ASK GunList Next(Gun);
   END WHILE;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" RecManvGranted TIME = ");
      ASK EventLog TO WriteReal(SimTime(),8,2);
      IF AllGunsClear
         ASK EventLog TO WriteLnStr(" AllGunsClear= TRUE");
      ELSE
         ASK EventLog TO WriteLnStr(" AllGunsClear= FALSE");
      END IF;
   END IF;
   IF AllGunsClear
{
      IF LogEvents
         ASK EventLog TO WriteLnStr(ID+" CTM Firing");
      END IF;
}
      TELL ClearedToManeuver TO Trigger;
   END IF;

END METHOD;


{-------------------------------------------------------------------------}

ASK METHOD HaltManeuvers;

VAR
   Gun : GunObj;

BEGIN
   StopManeuvering := TRUE;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" halt MANEUVERS SIMTIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
END METHOD;


{-------------------------------------------------------------------------}

ASK METHOD DropAll;

VAR
   Gun : GunObj;

BEGIN
   Process := pIdle;
   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ
      ASK Gun TO DropAll;
```

```
   END WHILE;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD RegistrationFire;

VAR
   Gun : GunObj;

BEGIN
   Process := pRegistration;
   Gun := ASK GunList First();
   WHILE (Gun <> NILOBJ)
      IF Gun.Status = Idle
         TELL Gun TO RegistrationFire;
         Gun := NILOBJ;
      ELSE
         Gun := ASK GunList Next(Gun);
         IF Gun = NILOBJ
            Process := pAwaitingRegistration;
         END IF;
      END IF;
   END WHILE;

END METHOD;

{-------------------------------------------------------------------------}


ASK METHOD EngageTarget(IN Target : TargetObj);

VAR
   Gun : GunObj;
   Assigned : BOOLEAN;

BEGIN

   IF LogEvents
      ASK EventLog TO WriteLnStr("START SHIP ENGAGE TARGET SHIPID = "+ID);
   END IF;
   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ
      IF ASK Gun Status = Idle
         ASK Gun TO EngageTarget(Target);
         IF LogEvents
            ASK EventLog TO WriteString(Target.ID+
                " assigned to "+Gun.ID+" at simtime = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
         Gun := NILOBJ;
         Assigned := TRUE;
      ELSE
         Gun := ASK GunList Next(Gun);
      END IF;
   END WHILE;

   IF NOT Assigned
```

```
      OUTPUT(ID," was told to engage a target when it had no guns available");
      IF LogEvents
         ASK EventLog TO WriteLnStr(ID+
          " was told to engage a target when it had no guns available");
      END IF;
   END IF;


END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD Maneuver(IN ManOrder : ManeuverRecord);

CONST
   DegRad = pi/180.0;

VAR
   Gun : GunObj;

BEGIN
   ManeuverStatus := AwaitingManeuver;
   WAIT DURATION ManOrder.DTime
   ON INTERRUPT
      ManeuverStatus := NoneScheduled;
      TERMINATE;
   END WAIT;
   IF StopManeuvering
      ManeuverStatus := NoneScheduled;
      TERMINATE;
   END IF;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" REQUESTING A MANEUVER TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ
      TELL Gun RequestManeuver;
      Gun := ASK GunList Next(Gun);
   END WHILE;
   ManeuverStatus := WaitingForCTMt;
   WAIT FOR ClearedToManeuver TO Fire;
   ON INTERRUPT
      ManeuverStatus := NoneScheduled;
      Gun := ASK GunList First();
      WHILE Gun <> NILOBJ
         ASK Gun TO CancelManeuver;
         Gun := ASK GunList Next(Gun);
      END WHILE;
      TERMINATE;
   END WAIT;
   ManeuverStatus := Maneuvering;
   ASK SELF TO UpdatePosition;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" COMMENCING MANEUVER TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
      ASK EventLog TO WriteString("   POSITION     x    = ");
      ASK EventLog TO WriteReal(LastLocation.x,8,2);
```

```
      ASK EventLog TO WriteString("        y        = ");
      ASK EventLog TO WriteLnReal(LastLocation.y,8,2);
      ASK EventLog TO WriteString("   beginning COURSE = ");
      ASK EventLog TO WriteReal(Course,8,2);
      ASK EventLog TO WriteString(" beginning speed = ");
      ASK EventLog TO WriteLnReal(Speed,8,2);
   END IF;
   LastLocation.x := LastLocation.x + SIN(Course*DegRad)*ManOrder.Advance
                     + COS(Course*DegRad)*ManOrder.Transfer;
   LastLocation.y := LastLocation.y + COS(Course*DegRad)*ManOrder.Advance
                     - SIN(Course*DegRad)*ManOrder.Transfer;
   Course := Course + ManOrder.DCourse;
   IF Course >= 360.0
      Course := Course - 360.0;
   ELSIF Course <= 0.0
      Course := Course + 360.0;
   END IF;
   Speed := Speed + ManOrder.DSpeed;
   IF Speed <= 0.0
      Speed := 0.0;
   END IF;
   WAIT DURATION ManOrder.Duration
   ON INTERRUPT
      StopManeuvering := TRUE;
   END WAIT;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" COMPLETED MANEUVER TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
      ASK EventLog TO WriteString("   POSITION  x  = ");
      ASK EventLog TO WriteReal(LastLocation.x,8,2);
      ASK EventLog TO WriteString("       y       = ");
      ASK EventLog TO WriteLnReal(LastLocation.y,8,2);
      ASK EventLog TO WriteString("   final COURSE = ");
      ASK EventLog TO WriteReal(Course,8,2);
      ASK EventLog TO WriteString(" final speed = ");
      ASK EventLog TO WriteLnReal(Speed,8,2);
   END IF;
   TimeLastLocation := SimTime();
   Gun := ASK GunList First();
   WHILE Gun <> NILOBJ
      ASK Gun TO CancelManeuver;
      Gun := ASK GunList Next(Gun);
   END WHILE;
   IF NOT StopManeuvering
      NextManeuver := NextManeuver.NextManeuver;
      IF (NextManeuver <> NILREC)
         TELL SELF TO Maneuver(NextManeuver);
         ManeuverStatus := AwaitingManeuver;
      ELSE
         ManeuverStatus := NoneScheduled;
      END IF;
   END IF;
END METHOD;

END OBJECT;

END MODULE.
```

```
9/11/91
DEFINITION MODULE GUN;

(*------------------------------------------------------------------------

MODULE NAME:     GUN
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/23/91
LAST MODIFIED:   8/1/91

DESCRIPTION:

This is a module of the NGFS simulation that defines a Gun located aboard a
ship.  A gun can be told to engage a target, register fire, or suspend
operations to allow a manuever.

------------------------------------------------------------------------*)
FROM MGrpMod IMPORT IDObj, ComponentListObj;
FROM GLOBALS IMPORT xyPoint;
FROM SHELL   IMPORT ShellObj;
FROM TARGET  IMPORT TargetObj, ALL TargetStatusTYPE, ALL SpotKindTYPE;
FROM MIOMod  IMPORT MStreamObj;
FROM IOMod   IMPORT FileUseType(Input);
FROM StatMod IMPORT SREAL;
FROM SEED    IMPORT MRandomObj;
FROM SimMod  IMPORT TriggerObj;

TYPE
   GunStatusTYPE = (Idle, Busy, Misfire, BrokeSoft, BrokeHard);
   GunProcessTYPE = (pIdle, pRegistration, pFFE, pSpotFire);
   WaitReasonTYPE = (Duration, SpotComms, Firing, Maneuver, None);

   MagazineTYPE = RECORD;
      ShellType : STRING;
      ShellPtr  : ShellObj;
      Next      : MagazineTYPE;
   END RECORD;

   GunObj = OBJECT(IDObj)

      Type               : STRING;
      Status             : GunStatusTYPE;
      Hot                : BOOLEAN;
      Process            : GunProcessTYPE;
      Assigned           : BOOLEAN;
      EnableRepair       : BOOLEAN;
      RepairTrigger      : TriggerObj;
      AlreadyTriggeredRepair : BOOLEAN;
      SpotGood           : BOOLEAN;
      WaitReason         : WaitReasonTYPE;
      ManeuverRequest    : BOOLEAN;
      ClearedToManeuver  : BOOLEAN;
      SpotterComms       : TriggerObj;
      ManeuverComplete   : TriggerObj;
      RequestDropAll     : BOOLEAN;
      SpotKind           : SpotKindTYPE;
      FOV                : INTEGER;
      Target             : TargetObj;
```

```
        Aimpoint            : xyPoint;
        LongAimpoint        : xyPoint;
        ShortAimpoint       : xyPoint;
        TOF                 : REAL;
        NumberFFERounds     : INTEGER;
        ShotsFired          : INTEGER;
        FiringRate          : SREAL;
        AveFiringRate       : SREAL;
        CycleTime           : REAL;
        MaxFlightTime       : REAL;
        ShipPtr             : ANYOBJ;
        NumRegRounds        : INTEGER;
        RegRange            : REAL;
        RegShell            : STRING;
        RegRoundsTracked    : INTEGER;
        PreferredShell      : ShellObj;
        ShotsBeforeHot      : INTEGER;
        Sigma               : xyPoint;
        RangeFactor         : REAL;
        Magazine            : MagazineTYPE;
        NumberOfComponents: INTEGER;
        ComponentList       : ComponentListObj;
        RandomGen           : MRandomObj;

        ASK METHOD CreateGun(IN GunType   : STRING;
                             IN InShipPtr : ANYOBJ;
                             IN InGunNum  : INTEGER);
        ASK METHOD InitializeGun;
        ASK METHOD ReceiveBDA(IN HitThisFFE   : BOOLEAN;
                              IN TargetStatus : TargetStatusTYPE);
        ASK METHOD ReceiveSpotResult(IN SpotResult : BOOLEAN;
                                     IN Correction : xyPoint);
        ASK METHOD TrackRound(IN ImpactPoint : xyPoint);
        ASK METHOD CancelManeuver;
        ASK METHOD EngageTarget(IN InTarget : TargetObj);
        ASK METHOD TermPrep(IN InStatus : GunStatusTYPE;
                            IN ReAssignTargets : BOOLEAN;
                            IN InProcess : GunProcessTYPE);
        ASK METHOD DropAll;
        ASK METHOD CalRunStats(IN  Rep                : INTEGER;
                               IN  StoppingPercentage : REAL;
                               IN  InvNormalCI        : REAL;
                               OUT RPM                : REAL;
                               OUT GunDone            : BOOLEAN);


        TELL METHOD Fire;
        TELL METHOD FFE;
        TELL METHOD SpotFire;
        TELL METHOD RegistrationFire;
        TELL METHOD RequestManeuver;
        TELL METHOD Repair;

END OBJECT;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE GUN;

{-------------------------------------------------------------------------

MODULE NAME:    GUN
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/24/91
LAST MODIFIED:  9/1/91

DESCRIPTION:

This is a module of the NGFS simulation that implements a Gun located
aboard a ship.  A gun can be told to engage a target, register fire, or
suspend operations to allow a maneuver.

-------------------------------------------------------------------------}

FROM SimMod  IMPORT Interrupt, SimTime;
FROM GLOBALS IMPORT xyPoint, TooLong, origin, ShellList, Distance, SpotDeltaT,
                    EventLog, LogEvents;
FROM TARGET  IMPORT TargetObj, ALL SpotKindTYPE, ALL TargetStatusTYPE;
FROM MGrpMod IMPORT ComponentListObj, ComponentObj;
FROM MIOMod  IMPORT MStreamObj, InputLog;
FROM IOMod   IMPORT FileUseType(Input);
FROM SHELL   IMPORT ShellObj, RangeFunction;
FROM SHIP    IMPORT ShipObj, ALL EngagementStatusTYPE;
FROM StarMod IMPORT RStatObj;
FROM MathMod IMPORT SQRT;

{-------------------------------------------------------------------------}



OBJECT GunObj;

ASK METHOD CreateGun(IN GunType   : STRING;
                     IN InShipPtr : ANYOBJ;
                     IN InGunNum  : INTEGER);

VAR
   GunFile    : MStreamObj;
   Ship       : ShipObj;
   Shell      : ShellObj;
   ShellKind  : STRING;
   MagShell   : MagazineTYPE;
   NumShellsInMag : INTEGER;
   I          : INTEGER;
   FailProb   : REAL;
   MTTRC      : REAL;
   MTTRH      : REAL;
   Description : STRING;
   Component  : ComponentObj;


BEGIN
   Type := GunType;
   ShipPtr := InShipPtr;
```

```
Ship := ShipPtr;
ID := Ship.ID;
INSERT(ID,10,"GUN-");
INSERT(ID,14,INTTOSTR(InGunNum));

NEW(GunFile);
ASK GunFile TO Open(GunType+".GUN",Input);
ASK GunFile TO SkipLines(9);
ASK GunFile TO ReadLnRealLOG(Sigma.x,"Sigma.x");
ASK GunFile TO ReadLnRealLOG(Sigma.y,"Sigma.y");
ASK GunFile TO ReadLnRealLOG(CycleTime,"CycleTime");
ASK GunFile TO ReadLnRealLOG(MaxFlightTime,"MaxFlightTime");
ASK GunFile TO ReadLnIntLOG(NumRegRounds,"NumRegRounds");
ASK GunFile TO ReadLnRealLOG(RegRange,"RegRange");
ASK GunFile TO ReadLnIntLOG(ShotsBeforeHot,"ShotsBeforeHot");
ASK GunFile TO ReadLnRealLOG(RangeFactor,"RangeFactor");
NEW(RandomGen);
ASK RandomGen TO GetSeed("Gun");
ASK InputLog TO WriteInt(RandomGen.originalSeed,10);
ASK InputLog TO WriteLnStr("    Nav error random number seed");
ASK GunFile TO SkipLines(1);
ASK GunFile TO ReadLnInt(NumShellsInMag);
ASK GunFile TO SkipLines(3);
NEW(MagShell);
Magazine := MagShell;
ASK GunFile TO ReadString(ShellKind);
ASK GunFile TO ReadString(MagShell.ShellType);
ASK GunFile TO SkipLines(1);
MagShell.ShellPtr := ASK ShellList PtrTo(ShellKind, FALSE);
IF MagShell.ShellPtr = NILOBJ
   NEW(Shell);
   ASK Shell TO CreateShell(ShellKind);
   ASK ShellList TO Add(Shell);
   MagShell.ShellPtr := Shell;
END IF;

FOR I := 2 TO NumShellsInMag
   NEW(MagShell.Next);
   MagShell := MagShell.Next;
   ASK GunFile TO ReadString(ShellKind);
   ASK GunFile TO ReadString(MagShell.ShellType);
   ASK GunFile TO SkipLines(1);
   MagShell.ShellPtr := ASK ShellList PtrTo(ShellKind, FALSE);
   IF MagShell.ShellPtr = NILOBJ
      NEW(Shell);
      ASK Shell TO CreateShell(ShellKind);
      ASK ShellList TO Add(Shell);
      MagShell.ShellPtr := Shell;
   END IF;
END FOR;
ASK GunFile TO SkipLines(1);
ASK InputLog TO WriteLn;
ASK GunFile TO ReadLnIntLOG(NumberOfComponents,"Number of Gun Components");
ASK InputLog TO WriteLn;
ASK GunFile TO SkipLines(3);
NEW(ComponentList);
ASK InputLog TO WriteLnStr(
              "  FailProb     MTTRC     MTTRH     Seed       Description");
```

```
   FOR I := 1 TO NumberOfComponents
      ASK GunFile TO ReadReal(FailProb);
      ASK InputLog TO WriteReal(FailProb,10,4);
      ASK GunFile TO ReadReal(MTTRC);
      ASK InputLog TO WriteReal(MTTRC,10,2);
      ASK GunFile TO ReadReal(MTTRH);
      ASK InputLog TO WriteReal(MTTRH,10,2);
      ASK GunFile TO ReadLine(Description);
      NEW(Component);
      ASK Component TO CreateComponent(Description,FailProb, MTTRC,
                                       MTTRH);
      ASK Component.RandomGen TO GetSeed("Gun");
      ASK InputLog TO WriteInt(ASK Component RandomGen.originalSeed,12);
      ASK InputLog TO WriteLnStr(Description);
      ASK ComponentList TO Add(Component);
   END FOR;
   NEW(SpotterComms);
   NEW(ManeuverComplete);
   NEW(RepairTrigger);
   ASK GunFile TO Close;
   DISPOSE(GunFile);


END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD InitializeGun;

BEGIN

   Status := Idle;
   Process := pIdle;
   Hot := FALSE;
   RequestDropAll := FALSE;
   Assigned := FALSE;
   EnableRepair := FALSE;
   SpotGood := FALSE;
   SpotKind := Bracket;
   WaitReason := None;
   ManeuverRequest := FALSE;
   ClearedToManeuver := FALSE;
   FOV := 0;
   Target := NILOBJ;
   Aimpoint := origin;
   LongAimpoint := origin;
   ShortAimpoint := origin;
   TOF := 0.0;
   ShotsFired := 0;
   RegRoundsTracked := 0;
   PreferredShell := NILOBJ;
   ASK GETMONITOR(FiringRate,RStatObj) TO Reset;

END METHOD;


{-------------------------------------------------------------------------}
```

```
ASK METHOD ReceiveBDA(IN HitThisFFE     : BOOLEAN;
                      IN TargetStatus   : TargetStatusTYPE);

BEGIN
   SpotGood := HitThisFFE;
   TELL SpotterComms TO Trigger;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD ReceiveSpotResult(IN SpotResult : BOOLEAN;
                             IN Correction : xyPoint);

BEGIN
   SpotGood := SpotResult;
   IF NOT SpotGood
      Aimpoint.x := Aimpoint.x + Correction.x;
      Aimpoint.y := Aimpoint.y + Correction.y;
   END IF;
   TELL SpotterComms TO Trigger;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD TrackRound(IN ImpactPoint : xyPoint);

BEGIN

   LongAimpoint.x := LongAimpoint.x + ImpactPoint.x;
   LongAimpoint.y := LongAimpoint.y + ImpactPoint.y;
   RegRoundsTracked := RegRoundsTracked + 1;

END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD CancelManeuver;

BEGIN
   ManeuverRequest := FALSE;
   ClearedToManeuver := FALSE;
   TELL ManeuverComplete TO Trigger;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" cancel maneuver TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD EngageTarget(IN InTarget : TargetObj);

VAR
   MagShell    : MagazineTYPE;
   ErrMsg      : STRING;
   PreferredShellType : STRING;
```

```
   Ship       : ShipObj;

BEGIN
   Ship := ShipPtr;
   Status := Busy;
   Target := InTarget;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" begin GUN ENGAGE TARGET TIME = ");
      ASK EventLog TO WriteReal(SimTime(),8,2);
      ASK EventLog TO WriteLnStr(" target = "+ Target.ID);
   END IF;
   ASK Ship TO UpdateEngagementStatus(FALSE);
   ASK Target TO UpdateAssigned(TRUE, SELF);
   PreferredShellType := Target.PreferredShell;
   MagShell := Magazine;
   WHILE MagShell.ShellType <> PreferredShellType
      MagShell := MagShell.Next;
      IF MagShell = NILREC
         OUTPUT(Target.ID," PreferredShellType ",PreferredShellType," not in ",
             "magazine of gun ",ID);
         IF LogEvents
            ASK EventLog TO WriteLnStr(Target.ID+" PreferredShellType "+
                      PreferredShellType+" not in magazine of gun "+ID);
         END IF;
         MagShell := Magazine;
         PreferredShellType := MagShell.ShellType;
         OUTPUT("   Substituting ",PreferredShellType);
         IF LogEvents
            ASK EventLog TO WriteLnStr("   Substituting "+PreferredShellType);
         END IF;
      END IF;
   END WHILE;
   PreferredShell := MagShell.ShellPtr;
   SpotKind := Target.NormalSpotKind;

   TELL SELF TO SpotFire;

END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD TermPrep(IN InStatus : GunStatusTYPE;
                    IN ReAssignTargets : BOOLEAN;
                    IN InProcess : GunProcessTYPE);
VAR
   Ship : ShipObj;
   TempTarget : TargetObj;

BEGIN
   Ship := ShipPtr;
   RequestDropAll := FALSE;
   TempTarget := Target;
   Process := pIdle;
   IF Target <> NILOBJ                    { no target if registration fire }
      ASK Target TO UpdateAssigned(FALSE,NILOBJ);
   END IF;Status := InStatus;
   Target := NILOBJ;
   IF ((InProcess = pFFE) OR (InProcess = pSpotFire)) AND
```

```
                                        (Ship.EngagementStatus < FullyEngaged)
          ASK Ship TO EngageTarget(TempTarget);
       END IF;
       IF ManeuverRequest
          IF NOT ClearedToManeuver
             ClearedToManeuver := TRUE;
             ASK Ship TO RecManvGranted;
          END IF;
       END IF;

       IF ((ASK RepairTrigger NumWaiting()) <> 0) AND
                                                 (NOT AlreadyTriggeredRepair)
          TELL RepairTrigger TO Trigger;
          AlreadyTriggeredRepair := TRUE;
        ELSE
           EnableRepair := TRUE;
        END IF;
        ASK Ship TO UpdateEngagementStatus(ReAssignTargets);

   END METHOD;

   {-------------------------------------------------------------------------}

   ASK METHOD DropAll;

   BEGIN
      IF WaitReason = Firing
         RequestDropAll := TRUE;
      ELSIF ManeuverRequest
         ASK ManeuverComplete TO InterruptTrigger;
      ELSIF WaitReason = SpotComms
         ASK SpotterComms TO InterruptTrigger;
      ELSE
         CASE Process
         WHEN pFFE:
            Interrupt(SELF,"FFE");
         WHEN pSpotFire:
            Interrupt(SELF,"SpotFire");
         OTHERWISE
            { do nothing }
         END CASE;
      END IF;
   END METHOD;

   {-------------------------------------------------------------------------}

   ASK METHOD CalRunStats(IN  Rep                : INTEGER;
                          IN  StoppingPercentage : REAL;
                          IN  InvNormalCI        : REAL;
                          OUT RPM                : REAL;
                          OUT GunDone            : BOOLEAN);

   BEGIN

      IF Status >= BrokeSoft
         FiringRate := FLOAT(ShotsFired)/SimTime();
      ELSIF TOF <> 0.0
         FiringRate := FLOAT(ShotsFired)/TOF;
```

```
   ELSE
      FiringRate := 0.0;
   END IF;
   RPM := FiringRate;
   AveFiringRate := FiringRate;
   IF Rep >= 10
      GunDone := ((StoppingPercentage*ASK GETMONITOR(AveFiringRate,RStatObj)
                 Mean()) >= (2.0*InvNormalCI*
                 (ASK GETMONITOR(AveFiringRate,RStatObj) StdDev()/
                 SQRT(FLOAT(Rep)-1.0))));
   ELSE
      GunDone := FALSE;
   END IF;

END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD Fire;

VAR
   Shell          : ShellObj;
   Ship           : ShipObj;
   Failure        : BOOLEAN;
   ActualAimpoint : xyPoint;
   MTTR           : REAL;
   Description    : STRING;
   Range          : REAL;

BEGIN

   IF LogEvents
      ASK EventLog TO WriteString(ID+" JUST FIRED AT ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   Ship := ShipPtr;
   ShotsFired := ShotsFired + 1;
   Hot := ShotsFired > ShotsBeforeHot;
   ASK ComponentList TO SampleForFailure(Hot, Failure, MTTR, Description);
   IF NOT Failure
      Shell := CLONE(PreferredShell);
      ASK Shell.ComponentList TO SampleForFailure(Hot, Failure, MTTR,
                                                  Description);
      IF NOT Failure
         ASK Ship TO UpdatePosition;
         IF Process = pRegistration      {THIS IS A REGISTRATION ROUND}
            Range := RegRange;
         ELSE
            Range := Distance(ASK Ship LastLocation, ASK Target Location);
         END IF;
         ActualAimpoint.x := ASK RandomGen Normal(Aimpoint.x,
                                 RangeFunction(Sigma.x, -1.0, Range));
         ActualAimpoint.y := ASK RandomGen Normal(Aimpoint.y,
                                 RangeFunction(Sigma.y, -1.0, Range));
         TELL Shell TO FlyToTarget(Target, ActualAimpoint, Range, SELF);
      ELSE
         {collect stats on shell failure}
         DISPOSE(Shell);
```

```
      END IF;

   ELSE
      (collect stats on gun failure)
   END IF;
   WAIT DURATION CycleTime;
   END WAIT;
   TOF := SimTime();

   IF Failure
      IF (MTTR > 0.0) AND (MTTR <= TooLong)
         IF LogEvents
            ASK EventLog TO WriteString(ID+" FAILED SOFT TIME = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
         Status := BrokeSoft;
         TELL SELF TO Repair IN MTTR;
      ELSIF MTTR > TooLong
         IF LogEvents
            ASK EventLog TO WriteString(ID+" FAILED HARD TIME = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
         Status := BrokeHard;
         TELL SELF TO Repair IN MTTR;
      ELSE
         IF LogEvents
            ASK EventLog TO WriteString(ID+" MISFIRE TIME = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
         Status := Misfire;
      END IF;
      IF LogEvents
         ASK EventLog TO WriteString(ID+" MTTR = ");
         ASK EventLog TO WriteLnReal(MTTR,8,2);
      END IF;
      EnableRepair := FALSE;
      AlreadyTriggeredRepair := FALSE;
   END IF;

END METHOD;

{-------------------------------------------------------------------------------}

TELL METHOD FFE;

VAR
   TargetStatus  : TargetStatusTYPE;
   ComStartDelay : REAL;
   Ship          : ShipObj;
   NumRounds     : INTEGER;

BEGIN
   Ship := ShipPtr;
   Process := pFFE;
   NumRounds := Target.RndsPerFFE;
   IF LogEvents
      ASK EventLog TO WriteString("HELLO FROM FFE GUNID = "+ID+
         " ROUNDS TO FIRE = ");
```

```
   ASK EventLog TO WriteLnInt(NumRounds,5);
   ASK EventLog TO WriteString("FIRING AT "+Target.ID+" time = ");
   ASK EventLog TO WriteLnReal(SimTime(),8,2);
END IF;
IF ManeuverRequest
   IF NOT ClearedToManeuver
      WaitReason := Maneuver;
      ClearedToManeuver := TRUE;
      ASK Ship RecManvGranted;
   END IF;
   WAIT FOR ManeuverComplete TO Fire;
   END WAIT;
   ClearedToManeuver := FALSE;
   WaitReason := None;
  END IF;
ASK Target TO StandbyForFFE;
REPEAT
   NumRounds := NumRounds - 1;
   WaitReason := Firing;
   WAIT FOR SELF TO Fire
   END WAIT;
   WaitReason := None;
   IF RequestDropAll
      ASK SELF TO TermPrep(Idle, TRUE, pIdle);
      TERMINATE;
   END IF;
UNTIL (NumRounds = 0) OR (Status >= BrokeSoft);
ComStartDelay := TOF + MaxFlightTime - SimTime();
IF ComStartDelay > 0.0
   WaitReason := Duration;
   WAIT DURATION ComStartDelay
   ON INTERRUPT
      WaitReason := None;
      ASK SELF TO TermPrep(Idle,TRUE,pIdle);
      TERMINATE;
   END WAIT;
   WaitReason := None;
END IF;
TELL Target TO ReportBDA;
WaitReason := SpotComms;
WAIT FOR SpotterComms TO Fire
ON INTERRUPT
   WaitReason := None;
   ASK SELF TO TermPrep(Idle,TRUE,pIdle);
   TERMINATE;
END WAIT;
WaitReason := None;

IF (ASK RepairTrigger NumWaiting()) <> 0
   IF LogEvents
      ASK EventLog TO WriteString(ID+
         " FFE TRIGGERING REPAIR TRIGGER TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   TELL RepairTrigger TO Trigger;
   AlreadyTriggeredRepair := TRUE;
ELSE
   EnableRepair := TRUE;
```

```
   END IF;
   IF Target.Status = Destroyed
{
      IF LogEvents
         ASK EventLog TO WriteLnStr(ID+" FFE SAYS TARGET DESTROYED");
      END IF;
}
      IF Status <= Misfire
         Status := Idle;
      END IF;
      TermPrep(Status,TRUE,pIdle);
      TERMINATE;
   ELSE
      CASE Status
      WHEN BrokeHard:
         TermPrep(BrokeHard,TRUE,pFFE);
         TERMINATE;
      WHEN BrokeSoft:
         { do nothing }
      OTHERWISE       {?Idle,Busy,?Misfire}
         IF SpotGood
            TELL SELF TO FFE;
         ELSE
            SpotKind := ASK Target NormalSpotKind;
            TELL SELF TO SpotFire;
         END IF;
      END CASE;
   END IF;

   IF LogEvents
      ASK EventLog TO WriteString(ID+" END ffe TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

END METHOD;

{-------------------------------------------------------------------------------}

TELL METHOD SpotFire;

VAR
   Ship       : ShipObj;
   HaltFiring : BOOLEAN;
   ComStartDelay : REAL;

BEGIN
   IF LogEvents
      ASK EventLog TO WriteString(ID+" START OF SpotFire SIMTIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   Ship := ShipPtr;
   Status := Busy;
   Process := pSpotFire;
   Aimpoint := Ship.Bias;
   CASE SpotKind
   WHEN Bracket..BracketLong:
      IF LogEvents
         ASK EventLog TO WriteLnStr("BRACKET LONG");
```

```
   END IF;
   IF SpotGood AND (FOV = Target.FOV)
      SpotKind := Direct;
      TELL SELF TO SpotFire;
      TERMINATE;
   END IF;
      Aimpoint.y := Aimpoint.y + Target.ECR*Target.ECRLong;
WHEN BracketShort:
   IF LogEvents
      ASK EventLog TO WriteLnStr("BRACKET SHORT");
   END IF;
   Aimpoint.y := Aimpoint.y - Target.ECR*Target.ECRLong;
OTHERWISE
   IF LogEvents
      ASK EventLog TO WriteLnStr("DIRECT SPOT");
   END IF;
END CASE;
SpotGood := FALSE;

REPEAT
   IF ManeuverRequest
      IF NOT ClearedToManeuver
         WaitReason := Maneuver;
         ClearedToManeuver := TRUE;
         ASK Ship RecManvGranted;
      END IF;
      WAIT FOR ManeuverComplete TO Fire;
      END WAIT;
      ClearedToManeuver := FALSE;
      WaitReason := None;
   END IF;
   ASK Target TO StandbyForSpot(SpotKind);
   WaitReason := Firing;
   WAIT FOR SELF TO Fire;
   END WAIT;
   WaitReason := None;
   IF RequestDropAll
      ASK SELF TO TermPrep(Idle, TRUE, pIdle);
      TERMINATE;
   END IF;
   CASE Status
   WHEN Idle..Busy:
      ComStartDelay := TOF + MaxFlightTime - SimTime();
      IF ComStartDelay > 0.0
         WaitReason := Duration;
         WAIT DURATION ComStartDelay
         ON INTERRUPT
            WaitReason := None;
            ASK SELF TO TermPrep(Idle,TRUE,pIdle);
            TERMINATE;
         END WAIT;
         WaitReason := None;
      END IF;
      TELL Target TO ReportSpotResult;
      WaitReason := SpotComms;
      WAIT FOR SpotterComms TO Fire
      ON INTERRUPT
         WaitReason := None;
```

```
            ASK SELF TO TermPrep(Idle,TRUE,pIdle);
            TERMINATE;
         END WAIT;
         WaitReason := None;
         IF Target.Status = Destroyed
            TermPrep(Idle,TRUE,pIdle);
            TERMINATE;
         END IF;
      WHEN BrokeHard:
         IF (ASK RepairTrigger NumWaiting()) <> 0
{
            IF LogEvents
               ASK EventLog TO WriteString(ID+
               " REPAIR TRIGGERED BY SPOTFIRE TIME hard fail= ");
               ASK EventLog TO WriteLnReal(SimTime(),8,2);
            END IF;
}
            TELL RepairTrigger TO Trigger;
            AlreadyTriggeredRepair := TRUE;
         ELSE
            EnableRepair := TRUE;
         END IF;
         ASK SELF TO TermPrep(BrokeHard,TRUE,pSpotFire);
         TERMINATE;
      WHEN Misfire:
         { do nothing }
      WHEN BrokeSoft:
         IF (ASK RepairTrigger NumWaiting()) <> 0
{
            IF LogEvents
               ASK EventLog TO WriteString(ID+
               " REPAIR TRIGGERED BY SPOTFIRE TIME soft fail= ");
               ASK EventLog TO WriteLnReal(SimTime(),8,2);
            END IF;
}
            TELL RepairTrigger TO Trigger;
            AlreadyTriggeredRepair := TRUE;
         ELSE
            EnableRepair := TRUE;
         END IF;
         TERMINATE;
      END CASE;
   UNTIL SpotGood;

   CASE SpotKind
   WHEN Bracket..BracketLong:
      LongAimpoint := Aimpoint;
      SpotKind := BracketShort;
      TELL SELF TO SpotFire;
   WHEN BracketShort:
      ShortAimpoint := Aimpoint;
      Aimpoint.x := (LongAimpoint.x + ShortAimpoint.x)/2.0;
      Aimpoint.y := (LongAimpoint.y + ShortAimpoint.y)/2.0;
      SpotGood := TRUE;
      FOV := Target.FOV;
      TELL SELF TO FFE;
```

```
   WHEN Direct:
      SpotGood := TRUE;
      FOV := Target.FOV;
      TELL SELF TO FFE;
   END CASE;

   IF LogEvents
      ASK EventLog TO WriteString(ID+" END spotfire TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

END METHOD;


{-------------------------------------------------------------------------------}

TELL METHOD RegistrationFire;

VAR
   Ship : ShipObj;
   Bias : xyPoint;
   I    : INTEGER;
   Successful : BOOLEAN;
   ComStartDelay : REAL;

BEGIN

   IF LogEvents
      ASK EventLog TO WriteString(ID+" start gun registration TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   Ship := ShipPtr;
   Status := Busy;
   Process := pRegistration;
   Target := NILOBJ;
   ASK Ship TO UpdateEngagementStatus(FALSE);
   Aimpoint := Ship.NavError;
   PreferredShell := Magazine.ShellPtr;
   RegRoundsTracked := 0;
   I := 0;
   REPEAT
      IF ManeuverRequest
          IF NOT ClearedToManeuver
            WaitReason := Maneuver;
            ClearedToManeuver := TRUE;
            ASK Ship RecManvGranted;
         END IF;
         WAIT FOR ManeuverComplete TO Fire;
         END WAIT;
         ClearedToManeuver := FALSE;
         WaitReason := None;
      END IF;
      WaitReason := Firing;
      WAIT FOR SELF TO Fire;
      END WAIT;
      WaitReason := None;
      IF RequestDropAll
         ASK SELF TO TermPrep(Idle, TRUE, pIdle);
```

```
         TERMINATE;
      END IF;
      IF Status = Busy
         I:=I+1;
      END IF;
   UNTIL (I = NumRegRounds) OR (Status >= BrokeSoft);
   CASE Status
   WHEN Busy:
      ComStartDelay := TOF + 2.0*MaxFlightTime - SimTime();
      IF ComStartDelay > 0.0
         WaitReason := Duration;
         WAIT DURATION ComStartDelay
         ON INTERRUPT
            { too late to stop fall of shot so do nothing}
         END WAIT;
         WaitReason := None;
      END IF;
      IF (RegRoundsTracked >= ROUND(0.75*FLOAT(NumRegRounds))) AND
            (RegRoundsTracked > 0)
         Bias.x := Aimpoint.x - LongAimpoint.x/FLOAT(RegRoundsTracked);
         Bias.y := Aimpoint.y - LongAimpoint.y/FLOAT(RegRoundsTracked);
         Successful := TRUE;
      END IF;
      Status := Idle;
   OTHERWISE
      IF (ASK RepairTrigger NumWaiting()) <> 0
         TELL RepairTrigger TO Trigger;
{
         IF LogEvents
            ASK EventLog TO WriteString(ID+
               " REPAIR TRIGGERED BY Registration TIME = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
}
         AlreadyTriggeredRepair := TRUE;
      ELSE
         EnableRepair := TRUE;
      END IF;
   END CASE;

   LongAimpoint := origin;
   ASK SELF TO TermPrep(Status, FALSE, pIdle);
   ASK Ship TO ReceiveRegStatus(Successful, Bias, SELF);
   IF LogEvents
      ASK EventLog TO WriteString(ID+" END GUN registration TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD RequestManeuver;

VAR
   Ship : ShipObj;

BEGIN
```

```
   IF LogEvents
      ASK EventLog TO WriteString(ID+
         " HAS BEEN REQUESTED TO GRANT A MANEUVER AT SIM TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
{
      ASK EventLog TO WriteString("CURRENT STATUS IS ");
      ASK EventLog TO WriteLnInt(ORD(Status),5);
      ASK EventLog TO WriteString("CURRENT PROCESS IS ");
      ASK EventLog TO WriteLnInt(ORD(Process),5);
}
   END IF;
   Ship := ShipPtr;
   ManeuverRequest := TRUE;
   CASE Status
   WHEN Busy..Misfire:
      { do nothing }
   OTHERWISE
      ClearedToManeuver := TRUE;
      ASK Ship TO RecManvGranted;
      IF LogEvents
         ASK EventLog TO WriteString(ID+" GRANTS MANEUVER REQUEST time = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
   END CASE;

END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD Repair;

VAR
   Ship : ShipObj;

BEGIN

   Ship := ShipPtr;
   IF NOT EnableRepair
{
      IF LogEvents
         ASK EventLog TO WriteString(ID+" REPAIR BEAT PROCESS SIMTIME = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
}
      WAIT FOR RepairTrigger TO Fire;
{
         IF LogEvents
            ASK EventLog TO WriteString(ID+
               " PROCESS IS enabling RepairTrigger simtime = ");
            ASK EventLog TO WriteLnReal(SimTime(),8,2);
         END IF;
}
      END WAIT;

   END IF;
```

```
   IF LogEvents
      ASK EventLog TO WriteString(ID+" REPAIRED AT TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;
   IF ClearedToManeuver
{
      IF LogEvents
         ASK EventLog TO WriteString(ID+" WAITING FOR MANV COMPLETE TIME = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
}
      WAIT FOR ManeuverComplete TO Fire;
      END WAIT;
     END IF;
   EnableRepair := FALSE;

   CASE Process
   WHEN pFFE:
      IF LogEvents
         ASK EventLog TO WriteString(ID+" REPAIR CALLING AN FFE TIME = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
      Status := Busy;
      TELL SELF TO FFE;
   WHEN pSpotFire:
      IF LogEvents
         ASK EventLog TO WriteString(ID+" REPAIR CALLING AN SpotFire TIME = ");
         ASK EventLog TO WriteLnReal(SimTime(),8,2);
      END IF;
      Status := Busy;
      TELL SELF TO SpotFire;
   OTHERWISE
      Status := Idle;
      Process := pIdle;
      ASK Ship TO UpdateEngagementStatus(TRUE);
   END CASE;
   IF LogEvents
      ASK EventLog TO WriteString(ID+" End Gun REPAIR FFE TIME = ");
      ASK EventLog TO WriteLnReal(SimTime(),8,2);
   END IF;

END METHOD;

END OBJECT;

END MODULE.
```

```
9/11/91
DEFINITION MODULE TARGET;


(*-------------------------------------------------------------------------

MODULE NAME:     TARGET
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/24/91
LAST MODIFIED:   7/29/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the Target object.
Target is responsible for creating itself and impacting rounds fired at it.



-------------------------------------------------------------------------*)
FROM MGrpMod IMPORT  IDObj;
FROM GLOBALS IMPORT  xyPoint, SpecifiedShellRecord;


TYPE

   TargetStatusTYPE = (Operational,Damaged,Destroyed);
   SpotKindTYPE = (Bracket, BracketLong, BracketShort, Direct);

   DFRecordTYPE = RECORD;
      DF         : REAL;
      ShellType  : STRING;
      Next       : DFRecordTYPE;
   END RECORD;


   TargetObj = OBJECT(IDObj);

      Type                 : STRING;
      Status               : TargetStatusTYPE;
      Priority             : INTEGER;
      Assigned             : BOOLEAN;
      Location             : xyPoint;
      LifePoints           : REAL;
      DFList               : DFRecordTYPE;
      HitThisFFE           : BOOLEAN;
      PreferredShell       : STRING;
      SpecifiedShells      : SpecifiedShellRecord;
      RndsPerFFE           : INTEGER;
      NormalSpotKind       : SpotKindTYPE;
      FOV                  : INTEGER;
      HQPtr                : ANYOBJ;
      GunPtr               : ANYOBJ;
      SpotGood             : BOOLEAN;
      Correction           : xyPoint;
      FFE                  : BOOLEAN;
      ECR                  : REAL;
      ECRLong              : REAL;
      BoxLong              : REAL;
```

```
    BoxWide                 : REAL;
    CorrectionFactor        : REAL;
    SpotKind                : SpotKindTYPE;
    SpotTime                : REAL;

    ASK METHOD CreateTarget(IN TargetType         : STRING;
                            IN TargetID           : STRING;
                            IN InLocation         : xyPoint;
                            IN InPriority         : INTEGER;
                            IN InFOV              : INTEGER;
                            IN InSpecifiedShells  : SpecifiedShellRecord;
                            IN InHQPtr            : ANYOBJ);

    ASK METHOD InitializeTarget;
    ASK METHOD UpdateAssigned(IN AssignedStatus : BOOLEAN;
                              IN InGunPtr       : ANYOBJ);

    ASK METHOD StandbyForSpot(IN SpotType : SpotKindTYPE);
    ASK METHOD StandbyForFFE;

    TELL METHOD ReportSpotResult;
    TELL METHOD ReportBDA;
    TELL METHOD ImpactRound(IN Shell : ANYOBJ);

    PRIVATE
       rLifePoints : REAL;

  END OBJECT;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE TARGET;


{-----------------------------------------------------------------------------

MODULE NAME:     TARGET
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/24/91
LAST MODIFIED:   9/1/91

DESCRIPTION:

This is a module of the NGFS simulation that implements the Target object.
Target is responsible for creating itself and impacting rounds fired at it.



-----------------------------------------------------------------------------}
FROM GLOBALS IMPORT xyPoint, Distance, origin, SpecifiedShellRecord, LOG,
                    EventLog, LogEvents;
FROM MIOMod  IMPORT MStreamObj, InputLog;
FROM IOMod   IMPORT FileUseType(Input);
FROM GUN     IMPORT GunObj;
FROM SHELL   IMPORT ShellObj;
FROM Debug   IMPORT TraceStream;
FROM HQ      IMPORT HQObj;
FROM SimMod  IMPORT SimTime;


OBJECT TargetObj;


ASK METHOD CreateTarget(IN TargetType        : STRING;
                        IN TargetID          : STRING;
                        IN InLocation        : xyPoint;
                        IN InPriority        : INTEGER;
                        IN InFOV             : INTEGER;
                        IN InSpecifiedShells : SpecifiedShellRecord;
                        IN InHQPtr           : ANYOBJ);


VAR
   TargetFile : MStreamObj;
   SpotKindString : STRING;
   I          : INTEGER;
   DFRecord   : DFRecordTYPE;
   NumDFs     : INTEGER;
   tempstr    : STRING;

BEGIN

   HQPtr := InHQPtr;
   Type := TargetType;
   ID := TargetID;
   Location := InLocation;
   Priority := InPriority;
   FOV := InFOV;
```

```
   SpecifiedShells := InSpecifiedShells;
   NEW(TargetFile);
   ASK TargetFile TO Open(TargetType+".TGT",Input);
   ASK TargetFile TO SkipLines(9);
   ASK TargetFile TO ReadLnRealLOG(LifePoints,"LifePoints");
   ASK TargetFile TO ReadLnStrLOG(PreferredShell,"PreferredShell");
   ASK TargetFile TO ReadLnIntLOG(RndsPerFFE,"RndsPerFFE");
   ASK TargetFile TO ReadLnRea_LOG(ECR,"ECR");
   ASK TargetFile TO ReadLnRealLOG(ECRLong,"ECRLong");
   ASK TargetFile TO ReadLnRealLOG(BoxLong,"BoxLong");
   ASK TargetFile TO ReadLnRealLOG(BoxWide,"BoxWide");
   ASK TargetFile TO ReadLnRealLOG(CorrectionFactor,"CorrectionFactor");
   ASK TargetFile TO ReadLnStrLOG(SpotKindString,"SpotKind");
   IF SUBSTR(1,1,SpotKindString) = "D"
      NormalSpotKind := Direct;
   ELSE
      NormalSpotKind := Bracket;
   END IF;
   ASK TargetFile TO ReadLnRealLOG(SpotTime,"SpotTime");
   ASK TargetFile TO SkipLines(1);
   ASK TargetFile TO ReadLnIntLOG(NumDFs,"NumDFs");
   ASK InputLog TO WriteLn;
   ASK TargetFile TO SkipLines(3);
   NEW(DFList);
   DFRecord := DFList;
   ASK InputLog TO WriteLnStr("DamageFactor  Shell Type");
   ASK InputLog TO WriteLn;
   FOR I := 1 TO NumDFs
      ASK TargetFile TO ReadReal(DFRecord.DF);
      ASK InputLog TO WriteReal(DFRecord.DF,9,2);
      ASK TargetFile TO ReadLnStr(DFRecord.ShellType);
      tempstr := "                ";
      REPLACE(tempstr,15-STRLEN(DFRecord.ShellType),14,DFRecord.ShellType);
      ASK InputLog TO WriteLnStr(tempstr);
      NEW(DFRecord.Next);
      DFRecord := DFRecord.Next;
   END FOR;
   ASK InputLog TO WriteLn;
   DISPOSE(DFRecord.Next);
   DFRecord.Next := NILREC;
   ASK TargetFile TO Close;
   DISPOSE(TargetFile);
   rLifePoints := LifePoints;

END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD InitializeTarget;

BEGIN
   Status     := Operational;
   Assigned   := FALSE;
   GunPtr     := NILOBJ;
   LifePoints := rLifePoints;
   SpotGood   := FALSE;
   FFE        := FALSE;
END METHOD;
```

```
{-------------------------------------------------------------------------}

ASK METHOD UpdateAssigned(IN AssignedStatus : BOOLEAN;
                          IN InGunPtr       : ANYOBJ);

BEGIN
   Assigned := AssignedStatus;
   GunPtr   := InGunPtr;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD StandbyForSpot(IN SpotCmd : SpotKindTYPE);

BEGIN
   SpotGood := FALSE;
   SpotKind := SpotCmd;
   FFE      := FALSE;
END METHOD;

{-------------------------------------------------------------------------}

ASK METHOD StandbyForFFE;

BEGIN
   FFE := TRUE;
   HitThisFFE := FALSE;
END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD ReportSpotResult;

VAR
   Gun : GunObj;

BEGIN

   Gun := GunPtr;
   WAIT DURATION SpotTime
   END WAIT;
   ASK Gun TO ReceiveSpotResult(SpotGood, Correction);

END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD ReportBDA;

VAR
   Gun : GunObj;

BEGIN

   Gun := GunPtr;
   WAIT DURATION SpotTime
   END WAIT;
```

```
   ASK Gun TO ReceiveBDA(HitThisFFE, Status);

END METHOD;

{-------------------------------------------------------------------------}

TELL METHOD ImpactRound(IN ShellPtr : ANYOBJ);

VAR
   Shell         : ShellObj;
   Gun           : GunObj;
   HQ            : HQObj;
   MissDistance  : REAL;
   DamageRadius  : REAL;
   ImpactPoint   : xyPoint;
   Damage        : REAL;
   DFRecord      : DFRecordTYPE;
   Found         : BOOLEAN;

BEGIN
   Shell := ShellPtr;
   HQ    := HQPtr;
   ImpactPoint := ASK Shell ImpactPoint;
   Gun := GunPtr;
   MissDistance := Distance(origin, ImpactPoint);
   DamageRadius := ECR + Shell.EDR;
   IF MissDistance <= DamageRadius
      IF LifePoints > 0.0
         DFRecord := DFList;
         WHILE (DFRecord <> NILREC) AND NOT Found
            IF DFRecord.ShellType = Shell.Type
               Found := TRUE;
            ELSE
               DFRecord := DFRecord.Next;
            END IF;
         END WHILE;
         IF NOT Found
            DFRecord := DFList;                { use default factor }
         END IF;
         Damage := Shell.MaxDamage*DFRecord.DF*((DamageRadius-MissDistance)*
                   (DamageRadius-MissDistance))/(DamageRadius*DamageRadius);
         LifePoints := LifePoints - Damage;
         IF LifePoints <= 0.0
            LifePoints := 0.0;
            Status := Destroyed;
            IF LogEvents
               ASK EventLog TO WriteString("!!!!!!!!!!!!!!!!!!!!!! "+ID+
                  " DESTROYED!!!!!!!!!!!!!!!! SimTime = ");
               ASK EventLog TO WriteLnReal(SimTime(),8,2);
            END IF;
         ELSE
            IF LogEvents
               ASK EventLog TO WriteString(ID+" HIT!!! ECR = ");
               ASK EventLog TO WriteReal(ECR,8,2);
               ASK EventLog TO WriteString(" MISS DISTANCE = ");
               ASK EventLog TO WriteReal(MissDistance,8,2);
               ASK EventLog TO WriteString(" LifePoints = ");
               ASK EventLog TO WriteLnReal(LifePoints,8,2);
```

```
            END IF;
            Status := Damaged;
         END IF;
         ASK HQ TO UpdateTargetValue;
      ELSE
         IF LogEvents
            ASK EventLog TO WriteLnStr(ID+" ALREADY DESTROYED");
         END IF;
      END IF;
      HitThisFFE := TRUE;
   END IF;

   IF NOT FFE
      Correction.x := -ImpactPoint.x;
      IF LogEvents
         ASK EventLog TO WriteString("IMPACT POINT = ");
         ASK EventLog TO WriteReal(ASK Shell ImpactPoint.x,8,2);
         ASK EventLog TO WriteLnReal(ASK Shell ImpactPoint.y,9,2);
      END IF;
      CASE SpotKind
      WHEN Bracket..BracketLong:
         IF LogEvents
            ASK EventLog TO WriteLnStr("BRACKET LONG");
         END IF;
         IF (ABS(ImpactPoint.x) < ECR*BoxWide) AND
            (ABS(ImpactPoint.y - ECR*ECRLong) < ECR*BoxLong)
            SpotGood := TRUE;
            Correction := origin;
         ELSE
            Correction.y := ECR*ECRLong - ImpactPoint.y;
         END IF;
      WHEN BracketShort:
         IF LogEvents
            ASK EventLog TO WriteLnStr("BRACKET SHORT");
         END IF;
         IF (ABS(ImpactPoint.x) < ECR*BoxWide) AND
            (ABS(ImpactPoint.y + ECR*ECRLong) < ECR*BoxLong)
            SpotGood := TRUE;
            Correction := origin;
         ELSE
            Correction.y := -ECR*ECRLong - ImpactPoint.y;
         END IF;
      WHEN Direct:
         IF LogEvents
            ASK EventLog TO WriteLnStr("BRACKET DIRECT");
         END IF;
         IF MissDistance < ECR
            SpotGood := TRUE;
            Correction := origin;
         ELSE
            Correction.y := -ImpactPoint.y;
         END IF;
      END CASE;
      Correction.x := Correction.x*CorrectionFactor;
      Correction.y := Correction.y*CorrectionFactor;
      IF LogEvents
         IF SpotGood
            ASK EventLog TO WriteString("SPOT RESULT, correction = TRUE ");
```

```
            ELSE
               ASK EventLog TO WriteString("SPOT RESULT, correction = FALSE ");
            END IF;
            ASK EventLog TO WriteReal(Correction.x,8,2);
            ASK EventLog TO WriteLnReal(Correction.y,9,2);
         END IF;
      END IF;

      DISPOSE(Shell);

   END METHOD;

   END OBJECT;

   END MODULE.
```

```
9/11/91
DEFINITION MODULE SHELL;


{-------------------------------------------------------------------------

MODULE NAME:     SHELL
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/24/91
LAST MODIFIED:   8/2/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the Shell object.
A Shell flies to a target and determines its impact point and whether or not
it successfully fires and fuses.

-------------------------------------------------------------------------}
FROM MGrpMod IMPORT  IDObj;
FROM GLOBALS IMPORT  xyPoint;
FROM TARGET  IMPORT TargetObj;
FROM MGrpMod IMPORT ComponentListObj;
FROM SEED    IMPORT MRandomObj;
TYPE

ShellObj = OBJECT(IDObj);

   Type                : STRING;
   Aimpoint            : xyPoint;
   ImpactPoint         : xyPoint;
   Target              : TargetObj;
   GunID               : STRING;
   Sigma               : xyPoint;
   MaxDamage           : REAL;
   EDR                 : REAL;
   RangeFactor         : REAL;
   Velocity            : REAL;
   NumberOfComponents  : INTEGER;
   ComponentList       : ComponentListObj;
   RandomGen           : MRandomObj;

   ASK METHOD CreateShell(IN ShellType : STRING);
   TELL METHOD FlyToTarget(IN Target     : TargetObj;
                           IN InAimpoint : xyPoint;
                           IN Range      : REAL;
                           IN GunPtr     : ANYOBJ);

END OBJECT;


PROCEDURE RangeFunction(IN Sigma       : REAL;
                        IN RangeFactor : REAL;
                        IN Range       : REAL) : REAL;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE SHELL;


(-------------------------------------------------------------------------

MODULE NAME:    SHELL
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/24/91
LAST MODIFIED:  9/1/91

DESCRIPTION:

This is a module of the NGFS simulation that implements the Shell object.
A Shell flies to a target and determines its impact point and whether or not
it successfully fires and fuses.

-------------------------------------------------------------------------)
FROM MGrpMod IMPORT ComponentObj;
FROM GLOBALS IMPORT xyPoint, EventLog, LogEvents;
FROM MIOMod  IMPORT MStreamObj, InputLog;
FROM IOMod   IMPORT FileUseType(Input);
FROM TARGET  IMPORT TargetObj;
FROM GUN     IMPORT GunObj;

PROCEDURE RangeFunction(IN RSigma      : REAL;
                        IN RangeFactor : REAL;
                        IN Range       : REAL) : REAL;

BEGIN
   IF RangeFactor < 0.0
      RETURN RSigma;
   ELSE
      RETURN RSigma*RangeFactor*Range;
   END IF;
END PROCEDURE;

OBJECT ShellObj;

ASK METHOD CreateShell(IN ShellKind : STRING);

VAR
   I : INTEGER;
   ShellFile : MStreamObj;
   FailProb  : REAL;
   MTTRC     : REAL;
   MTTRH     : REAL;
   Description : STRING;
   Component   : ComponentObj;

BEGIN

   ID := ShellKind;
   NEW(ShellFile);
   ASK InputLog TO WriteLnStr(
                "**************************************************");
   ASK InputLog TO WriteLnStr(ShellKind);
   ASK InputLog TO WriteLnStr(
```

```
                "*************************************************");
   ASK ShellFile TO Open(ShellKind+".SHL",Input);
   ASK ShellFile TO SkipLines(9);
   ASK ShellFile TO ReadLnStrLOG(Type,"TYPE");
   ASK ShellFile TO ReadLnRealLOG(Sigma.x,"Sigma.x");
   ASK ShellFile TO ReadLnRealLOG(Sigma.y,"Sigma.y");
   ASK ShellFile TO ReadLnRealLOG(MaxDamage,"MaxDamage");
   ASK ShellFile TO ReadLnRealLOG(EDR,"EDR");
   ASK ShellFile TO ReadLnRealLOG(Velocity,"VELOCITY");
   ASK ShellFile TO ReadLnRealLOG(RangeFactor,"RangeFactor");
   NEW(RandomGen);
   ASK RandomGen TO GetSeed("Shell");
   ASK InputLog TO WriteInt(RandomGen.originalSeed,10);
   ASK InputLog TO WriteLnStr("    Shell error random number seed");
   ASK ShellFile TO SkipLines(1);
   ASK ShellFile TO ReadLnIntLOG(NumberOfComponents,"Number Of Shell "+
                                 "Components");
   ASK ShellFile TO SkipLines(3);
   NEW(ComponentList);
   ASK InputLog TO WriteLnStr(
                "  FailProb     MTTRC     MTTRH     Seed       Description");
   FOR I := 1 TO NumberOfComponents
      ASK ShellFile TO ReadReal(FailProb);
      ASK InputLog TO WriteReal(FailProb,10,4);
      ASK ShellFile TO ReadReal(MTTRC);
      ASK InputLog TO WriteReal(MTTRC,10,2);
      ASK ShellFile TO ReadReal(MTTRH);
      ASK InputLog TO WriteReal(MTTRH,10,2);
      ASK ShellFile TO ReadLine(Description);
      NEW(Component);
      ASK Component TO CreateComponent(Description, FailProb,
                                       MTTRC, MTTRH);
      ASK (ASK Component RandomGen) TO GetSeed("Shell");
      ASK InputLog TO WriteInt(ASK Component RandomGen.originalSeed,12);
      ASK InputLog TO WriteLnStr(Description);
      ASK ComponentList TO Add(Component);
   END FOR;
   ASK InputLog TO WriteLnStr(
                "*************************************************");

   ASK ShellFile TO Close;
   DISPOSE(ShellFile);

END METHOD;


{-------------------------------------------------------------------------}


TELL METHOD FlyToTarget(IN Target     : TargetObj;
                        IN InAimpoint : xyPoint;
                        IN Range      : REAL;
                        IN GunPtr     : ANYOBJ);


VAR
   Gun : GunObj;
```

```
BEGIN

   Aimpoint := InAimpoint;
   WAIT DURATION Range/Velocity;
   END WAIT;

   ImpactPoint.x := ASK RandomGen Normal(Aimpoint.x,
                                RangeFunction(Sigma.x,RangeFactor,Range));
   ImpactPoint.y := ASK RandomGen Normal(Aimpoint.y,
                                RangeFunction(Sigma.y,RangeFactor,Range));

   IF Target = NILOBJ                          {This is a Registration Round}
      Gun := GunPtr;
      ASK Gun TO TrackRound(ImpactPoint);
      DISPOSE(SELF);
   ELSE
      TELL Target TO ImpactRound(SELF);
   END IF;

END METHOD;

END OBJECT;

END MODULE.
```

```
9/11/91
DEFINITION MODULE SEED;

(--------------------------------------------------------------------------

MODULE NAME:     SEED
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    7/31/91
LAST MODIFIED:   8/19/91

DESCRIPTION:

This is a module that defines a SeederObj object and a MRandomObj as well as
the variable seeder.

The SeederObj is a random number generator seed manager that has two methods

ASK METHOD ReadSeeds(<seedfilename>)

This method reads a seed file of the following format into memory

<name> #1
 ##
 ##
 ##
<name2> #2
 ##
 ##
 ##
ENDFILE.

Where <name1> is the name of the first string of seeds and #1 is the number
of seeds in the first string of seeds and ## are the seeds
ENDFILE.  designates the end of the file.

ASK METHOD GetNextSeed(<seedlistname>,seed);

This method gets the next seed in the string of seedlistname seeds.

MRandomObj is a RandomObj that adds one method, GetSeed(<name>)

This method queries the SeederObj for the next seed from the <name> seed list


--------------------------------------------------------------------------)

FROM RandMod IMPORT RandomObj;

TYPE
   SeedListTYPE = RECORD

      Name          : STRING;
      Seeds         : ARRAY INTEGER OF INTEGER;
      NumberOfSeeds : INTEGER;
      NextSeed      : INTEGER;
      NextSeedList  : SeedListTYPE;

   END RECORD;
```

```
   SeederObj = OBJECT
      SeedList : SeedListTYPE;
      ASK METHOD ReadSeeds(IN SeedFileName : STRING);
      ASK METHOD GetNextSeed(IN  SeedListName : STRING;
                             OUT Seed         : INTEGER);
      ASK METHOD ObjTerminate;

   END OBJECT;

   MRandomObj = OBJECT(RandomObj);

      ASK METHOD GetSeed(IN SeedListName : STRING);

   END OBJECT;

VAR

   seeder : SeederObj;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE SEED;

(-------------------------------------------------------------------------

MODULE NAME:     SEED
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    7/31/91
LAST MODIFIED:   8/19/91

DESCRIPTION:

This is a module that implements a SeederObj object and a MRandomObj as well as
the variable seeder.

The SeederObj is a random number generator seed manager that has two methods

ASK METHOD ReadSeeds(<seedfilename>)

This method reads a seed file of the following format into memory

<name> #1
 ##
 ##
 ##
<name2> #2
 ##
 ##
 ##
ENDFILE.

Where <name1> is the name of the first string of seeds and #1 is the number
of seeds in the first string of seeds and ## are the seeds
ENDFILE.  designates the end of the file.

ASK METHOD GetNextSeed(<seedlistname>,seed);

This method gets the next seed in the string of seedlistname seeds.

MRandomObj is a RandomObj that adds one method, GetSeed(<name>)

This method queries the SeederObj for the next seed from the <name> seed list

-------------------------------------------------------------------------)

FROM RandMod IMPORT RandomObj;
FROM MIOMod  IMPORT MStreamObj, InputLog;
FROM IOMod   IMPORT FileUseType(Input);
FROM UtilMod IMPORT RuntimeError;

OBJECT SeederObj;

ASK METHOD ReadSeeds(IN SeedFileName : STRING);

VAR
   SeedFile     : MStreamObj;
   Done         : BOOLEAN;
   SeedListName : STRING;
```

```
   I            : INTEGER;
   TempSeedList : SeedListTYPE;

BEGIN
   NEW(SeedFile);
   ASK SeedFile TO Open(SeedFileName,Input);
   WHILE (NOT Done) AND (NOT SeedFile.eof)
      ASK SeedFile TO ReadString(SeedListName);
      IF SeedListName = "ENDFILE."
         Done := TRUE;
      ELSE
         NEW(SeedList);
         SeedList.Name := SeedListName;
         ASK SeedFile TO ReadLnInt(SeedList.NumberOfSeeds);
         NEW(SeedList.Seeds, 1..SeedList.NumberOfSeeds);
         SeedList.NextSeed := 1;
         FOR I := 1 TO SeedList.NumberOfSeeds
            ASK SeedFile TO ReadLnInt(SeedList.Seeds[I]);
         END FOR;
         SeedList.NextSeedList := TempSeedList;
         TempSeedList := SeedList;
      END IF;
   END WHILE;
   ASK SeedFile TO Close;
   DISPOSE(SeedFile);
{     TempSeedList := SeedList;
      WHILE TempSeedList <> NILREC
         OUTPUT(" SEED LIST DUMP FOR > ",TempSeedList.Name);
         OUTPUT(" NUMBER OF SEEDS = ",TempSeedList.NumberOfSeeds);
         FOR I := 1 TO TempSeedList.NumberOfSeeds
            OUTPUT(TempSeedList.Seeds[I]);
         END FOR;
         TempSeedList := TempSeedList.NextSeedList;
      END WHILE;
}
END METHOD;

ASK METHOD GetNextSeed(IN  SeedListName : STRING;
                       OUT Seed         : INTEGER);

VAR
   CurrentSeedList : SeedListTYPE;

BEGIN
   CurrentSeedList := SeedList;
   WHILE (CurrentSeedList <> NILREC) AND (CurrentSeedList.Name <> SeedListName)
      CurrentSeedList := CurrentSeedList.NextSeedList;
   END WHILE;
   IF CurrentSeedList = NILREC
      RuntimeError("NGFS ERROR: SeedListName >"+SeedListName+
                   "< not found in seeder");
   ELSE
      IF CurrentSeedList.NextSeed > CurrentSeedList.NumberOfSeeds
         RuntimeError("NGFS ERROR: Ran out of seeds in seed list >"+
                      SeedListName+"< Number in list = "+
                      INTTOSTR(CurrentSeedList.NumberOfSeeds));

      ELSE
```

```
         Seed := CurrentSeedList.Seeds[CurrentSeedList.NextSeed];
         CurrentSeedList.NextSeed := CurrentSeedList.NextSeed + 1;
      END IF;
   END IF;
END METHOD;

ASK METHOD ObjTerminate;

VAR
   TempSeedList : SeedListTYPE;

BEGIN
   WHILE SeedList <> NILREC
      DISPOSE(SeedList.Seeds);
      TempSeedList := SeedList.NextSeedList;
      DISPOSE(SeedList);
      SeedList := TempSeedList;
   END WHILE;
END METHOD;

END OBJECT;


OBJECT MRandomObj;

ASK METHOD GetSeed(IN SeedListName : STRING);

VAR
   seed : INTEGER;

BEGIN

   ASK seeder TO GetNextSeed(SeedListName, seed);
   ASK SELF TO SetSeed(seed);

END METHOD;

END OBJECT;

END MODULE.
```

```
9/11/91
DEFINITION MODULE GLOBALS;

(*-------------------------------------------------------------------------

MODULE NAME:    GLOBALS
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/29/91
LAST MODIFIED:  9/1/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the GLOBAL data
structures used in the program with the exception of ShellList which
is defined in MGrpMod;

-------------------------------------------------------------------------*)

FROM RandMod IMPORT RandomObj;
FROM MGrpMod IMPORT ListObj;
FROM MIOMod  IMPORT MStreamObj;

TYPE

xyPoint = FIXED RECORD
   x : REAL;
   y : REAL;
END RECORD;

SpecifiedShellRecord = RECORD
   ShellKind : STRING;
   NextShell : SpecifiedShellRecord;
END RECORD;


VAR
   ShellList  : ListObj;
   origin     : xyPoint;
   TooLong    : REAL;
   SpotDeltaT : REAL;
   LOG        : MStreamObj;
   LogEvents  : BOOLEAN;
   EventLog   : MStreamObj;

PROCEDURE Distance(IN location1 : xyPoint;
                   IN location2 : xyPoint): REAL;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE GLOBALS;

(*-------------------------------------------------------------------------

MODULE NAME:     GLOBALS
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/29/91
LAST MODIFIED:   5/2/91

DESCRIPTION:

This is a module of the NGFS simulation that implements the GLOBAL data
structures and procedures used in the program.

-------------------------------------------------------------------------*)

FROM RandMod IMPORT RandomObj;
FROM MathMod IMPORT SQRT;



PROCEDURE Distance(IN location1 : xyPoint;
                   IN location2 : xyPoint): REAL;

BEGIN

   RETURN SQRT((location1.x-location2.x)*(location1.x-location2.x)
               + (location1.y-location2.y)*(location1.y-location2.y));
END PROCEDURE;

END MODULE.
```

```
9/11/91
DEFINITION MODULE MIOMod;


{-------------------------------------------------------------------------

MODULE NAME:    MIOMod
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/16/91
LAST MODIFIED:  7/30/91

DESCRIPTION:

This is a module of the NGFS simulation that modifies the standard Stream.OBJ
of IOMod to shorten input and output.  It also defines a new type of file
allows indexing on input by adding a PositionOn method that positions the
file to the begining of a record based on the following file format:

BOF
11 lines of comments
INTEGER         (OFFSET)  offset to 1st line of 1st record skips over index
INTEGER         (RECORD LENGTH) length of each record
2 lines of comments
STRING          (TYPE)    index on these strings
STRING
.
.
.               I lines in index
2 lines of comments
first data line of record
.
.  record length of lines in each record
.
first data line of next record etc


-------------------------------------------------------------------------}


FROM IOMod   IMPORT StreamObj;
FROM UtilMod IMPORT RuntimeError;

TYPE

MStreamObj = OBJECT(StreamObj);

   ASK METHOD ReadLnInt(OUT n : INTEGER);
   ASK METHOD ReadLnReal(OUT x : REAL);
   ASK METHOD ReadLnStr(OUT str : STRING);
   ASK METHOD SkipLines(IN n : INTEGER);

   ASK METHOD WriteLnInt(IN num        : INTEGER;
                         IN fieldwidth : INTEGER);
   ASK METHOD WriteLnReal(IN num        : REAL;
                          IN fieldwidth : INTEGER;
                          IN precision  : INTEGER);
   ASK METHOD WriteLnStr(IN str : STRING);
   ASK METHOD WriteLines(IN n : INTEGER);
```

```
   ASK METHOD ReadLnIntLOG(OUT n : INTEGER;
                           IN  str : STRING);
   ASK METHOD ReadLnRealLOG(OUT x : REAL;
                            IN  str : STRING);
   ASK METHOD ReadLnStrLOG(OUT str   : STRING;
                             IN  instr : STRING);


END OBJECT;

IndexStreamObj = OBJECT(MStreamObj);

    ASK METHOD PositionOn(IN Type : STRING);

END OBJECT;

VAR
   InputLog : MStreamObj;

END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE MIOMod;

{-------------------------------------------------------------------------

MODULE NAME:     MIOMod
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/16/91
LAST MODIFIED:   8/1/91

DESCRIPTION:

This is a module of the NGFS simulation that modifies the standard Stream.OBJ
of IOMod to shorten input and output.

-------------------------------------------------------------------------}
FROM IOMod   IMPORT StreamObj;
FROM UtilMod IMPORT RuntimeError;

OBJECT MStreamObj;

ASK METHOD ReadLnInt(OUT n : INTEGER);

VAR
   str : STRING;

BEGIN

   ASK SELF TO ReadInt(n);
   ASK SELF TO ReadLine(str);

END METHOD;

ASK METHOD ReadLnReal(OUT x : REAL);

VAR
   str : STRING;

BEGIN

   ASK SELF TO ReadReal(x);
   ASK SELF TO ReadLine(str);

END METHOD;


ASK METHOD ReadLnStr(OUT str : STRING);

VAR
   strx : STRING;

BEGIN

   ASK SELF TO ReadString(str);
   ASK SELF TO ReadLine(strx);

END METHOD;
```

```
ASK METHOD SkipLines(IN n : INTEGER);

VAR
   str : STRING;
   I   : INTEGER;

BEGIN

   FOR I := 1 TO n;

      ASK SELF TO ReadLine(str);

   END FOR;

END METHOD;

ASK METHOD WriteLnInt(IN num, fieldwidth : INTEGER);

BEGIN

   ASK SELF TO WriteInt(num, fieldwidth);
   ASK SELF TO WriteLn;

END METHOD;

ASK METHOD WriteLnReal(IN num        : REAL;
                       IN fieldwidth : INTEGER;
                       IN precision  : INTEGER);

BEGIN

   ASK SELF TO WriteReal(num, fieldwidth,precision);
   ASK SELF TO WriteLn;

END METHOD;
ASK METHOD WriteLnStr(IN str : STRING);

BEGIN

   ASK SELF TO WriteString(str);
   ASK SELF TO WriteLn;

END METHOD;

ASK METHOD WriteLines(IN n : INTEGER);

VAR
   I : INTEGER;

BEGIN

   FOR I := 1 TO n
      ASK SELF TO WriteLn;
   END FOR;

END METHOD;
```

```
ASK METHOD ReadLnIntLOG(OUT n : INTEGER;
                        IN  str : STRING);
BEGIN
   ASK SELF TO ReadLnInt(n);
   ASK InputLog TO WriteInt(n,10);
   ASK InputLog TO WriteLnStr("    " + str);
END METHOD;

ASK METHOD ReadLnRealLOG(OUT x : REAL;
                         IN  str : STRING);
BEGIN
   ASK SELF TO ReadLnReal(x);
   ASK InputLog TO WriteReal(x,10,2);
   ASK InputLog TO WriteLnStr("    " + str);
END METHOD;

ASK METHOD ReadLnStrLOG(OUT str   : STRING;
                        IN  instr : STRING);
VAR
   tempstr : STRING;

BEGIN
   ASK SELF TO ReadLnStr(str);
   tempstr := "               ";
   IF STRLEN(str) > 10
      tempstr := str + "  ";
   ELSE
      REPLACE(tempstr,11 - STRLEN(str),10,str);
   END IF;
   ASK InputLog TO WriteLnStr(tempstr + instr);
END METHOD;

END OBJECT;

OBJECT IndexStreamObj;

ASK METHOD PositionOn(IN Type : STRING);

VAR
   Offset : INTEGER;
   RecordLength : INTEGER;
   Index        : INTEGER;
   Found        : BOOLEAN;
   MatchType    : STRING;
   ErrMsg       : STRING;

BEGIN
   ASK SELF TO SkipLines(11);
   ASK SELF TO ReadLnInt(Offset);
   ASK SELF TO ReadLnInt(RecordLength);
   ASK SELF TO SkipLines(4);

   REPEAT
      ReadLnStr(MatchType);
      Index := Index+1;
   UNTIL (MatchType = Type) OR (Index = Offset);

   IF (Index = Offset) AND (MatchType <> Type)
```

```
        ErrMsg := "INTERNAL NGFS ERROR --- following type not in file>";
        INSERT(ErrMsg,52,Type);
        RuntimeError(ErrMsg);
    ELSE
       ASK SELF TO SkipLines((Index-1)*RecordLength+Offset-Index);
    END IF;

END METHOD;


END OBJECT;
END MODULE.
```

```
9/11/91
DEFINITION MODULE MGrpMod;

{-------------------------------------------------------------------------

MODULE NAME:     MGrpMod
AUTHOR:          LT. RICHARD L. DARDEN
DATE WRITTEN:    4/16/91
LAST MODIFIED:   5/1/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the following objects
based on the QueueObj of GrpMod:

   ComponentListObj  :   Maintains a list of components that can fail of an
                         object and allows a querry of which ones fail

   ListObj           :   Maintains a list of objects and allows a querry of
                         a pointer to an object based on a string field ID


-------------------------------------------------------------------------}

FROM SEED IMPORT MRandomObj;
FROM GrpMod  IMPORT QueueObj;

TYPE

   IDObj = OBJECT

      ID : STRING;

   END OBJECT;

   ComponentObj = OBJECT(IDObj)

      FailProb :  REAL;
      MTTRC    :  REAL;
      MTTRH    :  REAL;
      RandomGen : MRandomObj;

      ASK METHOD CreateComponent(IN ID       : STRING;
                                 IN FailProb : REAL;
                                 IN MTTRC    : REAL;
                                 IN MTTRH    : REAL);

   END OBJECT;

   ComponentListObj = OBJECT(QueueObj)

      ASK METHOD SampleForFailure(IN   Hot         : BOOLEAN;
                                  OUT  Failure     : BOOLEAN;
                                  OUT  MTTR        : REAL;
                                  OUT  DESCRIPTION : STRING);
```

```
   END OBJECT;

   ListObj = OBJECT(QueueObj)

      ASK METHOD PtrTo(IN ID               : STRING;
                       IN HaltIfNotOnList : BOOLEAN) : ANYOBJ;

   END OBJECT;


END MODULE.
```

```
9/11/91
IMPLEMENTATION MODULE MGrpMod;


(-------------------------------------------------------------------------

MODULE NAME:    MGrpMod
AUTHOR:         LT. RICHARD L. DARDEN
DATE WRITTEN:   4/16/91
LAST MODIFIED:  5/1/91

DESCRIPTION:

This is a module of the NGFS simulation that defines the following objects
based on the QueueObj of GrpMod:

   ComponentListObj  :   Maintains a list of components that can fail of an
                         object and allows a querry of which ones fail

   ListObj           :   Maintains a list of objects and allows a querry of
                         a pointer to an object based on a string field ID


-------------------------------------------------------------------------)


FROM UtilMod IMPORT RuntimeError;


OBJECT ComponentObj;

ASK METHOD CreateComponent(IN InID       : STRING;
                           IN InFailProb : REAL;
                           IN InMTTRC    : REAL;
                           IN InMTTRH    : REAL);

BEGIN

   ID := InID;
   FailProb := InFailProb;
   MTTRC := InMTTRC;
   MTTRH := InMTTRH;

   NEW(RandomGen);

END METHOD;


END OBJECT;


OBJECT   ComponentListObj;

ASK METHOD SampleForFailure(IN   Hot         : BOOLEAN;
                            OUT  Failure     : BOOLEAN;
                            OUT  MTTR        : REAL;
                            OUT  Description : STRING);
```

```
VAR
   Component : ComponentObj;
   Sample    : REAL;

BEGIN

   Component := ASK SELF First();
   WHILE (Component <> NILOBJ) AND (NOT Failure);
      Sample := ASK Component.RandomGen UniformReal(0.0, 1.0);
      IF Sample <= Component.FailProb
         IF Hot
            MTTR := Component.MTTRH;
         ELSE
            MTTR := Component.MTTRC;
         END IF;
         Failure := TRUE;
         Description := Component.ID;
      ELSE
         Component := ASK SELF Next(Component);
      END IF;
   END WHILE;
END METHOD;

END OBJECT;

OBJECT ListObj;

ASK METHOD PtrTo(IN ID              : STRING;
                 IN HaltIfNotOnList : BOOLEAN) : ANYOBJ;

VAR

   Ptr    : IDObj;
   ErrMsg : STRING;

BEGIN

   Ptr := ASK SELF First();
   WHILE (Ptr <> NILOBJ) AND (ASK Ptr ID <> ID);
      Ptr := ASK SELF Next(Ptr);
   END WHILE;
   IF (Ptr = NILOBJ) AND HaltIfNotOnList
      ErrMsg := "Internal NGFS error.  This was not on the list=>";
      INSERT(ErrMsg,49,ID);
      RuntimeError(ErrMsg);
   ELSE
      RETURN Ptr;
   END IF;

END METHOD;

END OBJECT;
END MODULE.
```

APPENDIX B SAMPLE NGFS INPUT FILES

```
9/11/91
*****************************************************************************

                    NGFS Simulation Parameters DATA FILE
                              (SIMPARM.DAT)

DATA FILE NAME : SIMPARM.DAT
DESCRIPTION    : This file sets up the simulation parameters for the NGFS
                 Simulation


*****************************************************************************

Calculate StopMode       How the simulation stops (Calculate or NumReps)
1000      MaxRep         Stop the simulation after this number of runs
0.20      StoppingPercentage  When StopMode equal Calculated run until the
                         convidence interval is this percentage of the value
1.96      InvNormalCI    Inverse Normal Function for 5% CI
NGFSSEED.DAT SeedFile    name of the random number generator seed file
FALSE     LogEvents      if TRUE logs simulation events to INPUT.LOG file

*****************************************************************************
```

9/11/91

****

NGFS SCENARIO DATA FILE

BASE LINE TEST SCENARIO

ACTUAL DataFileName
Tue Aug 20 01:38:29 1991
Richard_Darden UpdatedBy

General DATA

10.0 TooLong If a gun is down for greater than this reassign tgt

TARGET DATA

12 NumTargets Number of Targets in Scenario

TargetType	TargetID	Xpos	Ypos	Priority	FOV	Specified Shells
T-72TANK	TANK__-01	8250.0	-2250.0	3	1	S5inHE1 x
T-72TANK	TANK__-02	8250.0	-4250.0	3	3	S5inHE1 x
T-72TANK	TANK__-03	7750.0	-6250.0	3	5	S5inHE1 x
T-72TANK	TANK__-04	7250.0	-8250.0	3	7	S5inHE1 x
BASICHQ	HQ____-01	11000.0	-6250.0	1	5	S5inHE x
AABAT	AABAT_-01	9500.0	-5250.0	2	4	none
AABAT	AABAT_-02	9500.0	-7250.0	2	6	none
ARTY	ARTY__-01	10000.0	-4250.0	4	3	none
ARTY	ARTY__-02	9000.0	-6250.0	4	6	none
INFANTRY	INF___-01	8250.0	-3250.0	5	2	none
INFANTRY	INF___-02	8000.0	-5250.0	5	4	none
INFANTRY	INF___-03	7500.0	-7250.0	5	6	none

SHIP DATA

4 NumShips Number of Ships in Scenario

ShipType	ShipID	Xpos	Ypos	Course	Speed	ManFileName
SPRUANCE	DD____-01	2000.0	-6000.0	000.0	5.0	Manv1.MAN
SPRUANCE	DD____-02	2000.0	-9000.0	000.0	5.0	Manv1.MAN
PERRY	FF____-01	2000.0	-7000.0	000.0	5.0	Manv1.MAN
PERRY	FF____-02	2000.0	-8000.0	000.0	5.0	Manv1.MAN

****

```
9/11/91
*****************************************************************************
                              NGFS SHIP DATA FILE

Spruance.SHP

SPRUANCE              ShipName
Mon Aug 19 23:44:44 1991
Richard_Darden        UpdatedBy

100.0           NavSigma.x    Navigation System Variance
100.0           NavSigma.y
2               NumGuns       Number of guns on board

   GunType      GunNumber

5in54cal            1
5in54cal            2

*****************************************************************************
```

9/11/91
****

NGFS GUN DATA FILE

5in54Cal

```
5in54cal        DataFileName
Mon Aug 19 23:44:44 1991
Richard_Darden UpdatedBy

2.0             Sigma.x         Gun Accuracy Variance
2.0             Sigma.y
0.05            CycleTime       Time required for gun to cycle
0.49            MaxFlightTime   Flight time for max range
4               NumRegRounds    Number of rounds in a registration fire
17000.0         RegRange        Registration range  YDS
50              ShotsBeforeHot  Number of Rounds before Gun is Hot
-1.0            RangeFactor     -1 to disable

5           Number of Shell types in Magazine

ShellKind      Type   Description

5inHE           HE    High explosive 5 in round
5inFRAG         FRAG  Fragmenting 5 in round
5inAP           AP    Armor Piercing 5 in round
5inHE           SUB   use HE round
5inHE           SPC   use HE round

49          Number of Gun Components

FailProb    MTTRC     MTTRH      DESCRIPTION

0.0004762     0.0       0.0
0.0010000     2.7       2.7
0.0010000    15.0      15.0
0.0004762     0.0       0.0
0.0005556     0.8       0.8
0.0012346     9.3       9.3
0.0031250     6.0       6.0
0.0012346     1.8       1.8
0.0066667    41.0      41.0
0.0025000     5.3       5.3
0.0006250     0.0       0.0
0.0006250     4.2       4.2
0.0000256     0.0       0.0
0.0000526     6.5       6.5
0.0008333     0.0       0.0
0.0008333     0.0       0.0
0.0008333     0.0       0.0
0.0008333     5.0       5.0
0.0008333     0.0       0.0
0.0000256     0.0       0.0
0.0000256    23.0      23.0
0.0000256     9.3       9.3
0.0000256     0.0       0.0
0.0000526     0.5       0.5
0.0000256     0.0       0.0
```

```
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     6.6       6.6
0.0003846     2.3       2.3
0.0001299     1.1       1.1
0.0000256     0.0       0.0
0.0000769     3.0       3.0
0.0000256     0.0       0.0
0.0000256     2.4       2.4
0.0006250     0.0       0.0
0.0000256     3.9       3.9
0.0001282     0.0       0.0
0.0000256     0.0       0.0
0.0000256     4.0       4.0
0.0000256     0.0       0.0
0.0000256    11.0      11.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0
0.0000256     0.0       0.0

*****************************************************************************
```

```
9/11/91
*******************************************************************************
                              SHELL DATA FILE  5inHE

5inHE   Data for a 5in High Explosive Shel

5inHE           DataFileName
Mon Aug 19 23:44:44 1991
Richard_Darden UpdatedBy

HE              ShellType     This is a High Explosive Shell
1.0             Sigma.x       Shell Accuracy Variance
1.0             Sigma.y
20.0            MaxDamage     Maximum damage this shell can cause
5.0             EDR           Effective Damage Radius
53149.6         Velocity      Shell velocity yds/MIN
-1.0            RangeFactor   coeficient of range accuracy -1 disabled

5               NumberOfComponents  number of components in shell that can
fail

FailProb    MTTRC     MTTRH      DESCRIPTION

 0.003       25.0      55.0      PRIMER
 0.00139     31.0      61.0      PROPELLENT CHARGE
 0.003       10.0      10.0      FIRING PIN
 0.005        0.0       0.0      EXPLOSIVE
 0.000083     0.0       0.0      DETONATOR

*******************************************************************************
```

```
9/11/91
*****************************************************************************
                              NGFS TARGET DATA FILE

BASICHQ          Contains data that simulates a BASIC HQ TARGET

BASICHQ              DataFileName
Mon Aug 19 23:44:44 1991
Richard_Darden UpdatedBy

35.0             LifePoints       This life points of this target
HE               PreferredShell   Shell type to use against this target
12               RndsPerFFE       Number of rounds per FFE for this target
10.0             ECR              Effective circular radius  yards
2.0              ECRLong          Offset to bracket boxes in *ECR units
1.0              BoxLong          Length of bracket box in *ECR units
1.0              BoxWide          Width of bracket box in *ECR units
0.9              CorrectionFactor  Only correct this much of miss distance
Bracket          SpotKind         Do bracket spotting on this target
0.50             SpotTime         time in seconds to spot

3                Number of Damage Factors listed

DamageFactor  Shell Type       Damage Factor of target to various shell types

    0.8            HE           <= (Default damage factor)
    0.8            FRAG
    0.4            AP

*****************************************************************************
```

9/11/91
****

NGFS MANUEVER DATA FILE #1
(MANV1.DAT)

DATA FILE NAME : MANV1.DAT
DESCRIPTION : 5000 YARD BATTLE LINE AT 5KTS INITIAL SPEED
LAST MODIFIED : 8/2/91

****

MANEUVER DATA

DeltaTime	DeltaCourse	DeltaSpeed	Advance	Transfer	Duration
0030.00	180.00	0.0	000.0	-1000.0	05.00

-1.0

****

```
9/11/91
Ship 50              > # SHIPS
  2017146438
  1413213671
   392053470
   684529876
  1007116790
   740250286
  1201661671
   370844991
  1969699122
  1708460431
   618976317
  1168596879
  1886201888
   392989881
   799298326
   246209260
   538370580
  2093539433
   320699838
  1372346883
   825278867
  1773135851
   839212284
   167499556
  1339809316
  1423630665
  2097690772
  1153906231
   488155557
  1929962608
   669496870
  1234903993
  1942562258
   689051767
  2000310426
  1134221203
   772777814
   871931336
  2127914512
  1136500243
  1252287221
   184365652
    73463703
    84061736
   471441434
  1506432110
   227315464
  1942719004
  1508165913
  1342117964
Gun 450              > # GUNS * ( 1 * NUMBER FAILURES PER GUN )
  1134221203
   772777814
   871931336
  2127914512
  1136500243
```

1252287221
184365652
73463703
84061736
471441434
1506432110
227315464
1942719004
1508165913
1342117964
2055753977
39581853
1453533614
483129542
467476811
516068878
965878699
128299071
985145439
619592575
1645390429
547070247
956650449
1823265775
1804951546
2147118575
325352551
1504174718
73055694
636998749
1462839730
28112438
951863662
404170188
1307113196
192162417
1013137689
871880609
591373335
777057464
273610459
1949859982
741836289
968440228
197592295
1813627466
1273220553
1157745798
26220005
619558279
536126172
1440331842
121184505
396451164
797548811
931571114
1471192250
362690338

1706360386
1564802464
1790797553
898350637
1279110304
443053755
1139079825
424828146
875272429
1362312062
947168028
2121235004
1506369355
1312461178
404767434
1639284818
614775439
890426103
1546747636
1231485493
1884505518
1285050892
1057831785
285229091
1532722664
75342124
507755637
1166570195
908409930
1404052414
2060078470
260299136
1975677990
1119262381
1185081855
1108652391
581362415
1214005104
1690068976
1575719926
1761483833
144642838
1345634196
241654488
1413242377
2112751716
1814690540
1457418367
1201189105
412509432
118831179
997138152
1972103304
753451007
813413829
387734608
1486611293
809921819

2146007969
402929923
1559707544
1040012126
831312807
1348540613
903104502
745518912
1269904814
241508321
1709501965
1504351011
1357221586
826302239
521235650
87021785
904974031
697964027
21976770
556862300
1598162604
1091760299
903105303
434022324
1885535037
622438816
422844916
870218640
1359038352
1059483416
926128229
829979802
1166525451
448874700
1280689831
1255985194
1911216000
76271663
1046574445
2132545692
922510944
1901633463
1663153658
1116780070
2017146438
1413213671
392053470
684529876
1007116790
740250286
1201661671
370844991
1969699122
1708460431
618976317
1168596879
1886201888
392989881

799298326
246209260
538370580
2093539433
320699838
1372346883
825278867
1773135851
839212284
167499556
1339809316
1423630665
2097690772
1153906231
488155557
1929962608
669496870
1234903993
1942562258
689051767
2000310426
2007288405
238816165
1266870678
1235626112
1091961447
717544387
890603925
1302083772
1124190290
938163111
1026962274
1949197326
2037584015
1779810405
1253972948
1810631854
2145226806
1570178581
754011695
1048507996
483116016
1095943763
810864707
2142874801
387941933
960279820
1442561861
447999152
2060725609
1408961130
1242956972
1405846315
1858555148
1404582979
2090418009
2052451294
740452247

1165117153
2087000425
1802999840
1322497268
1462127324
1039959873
1580771462
 541636181
1149282773
1532201206
1252835522
 360144085
 969638136
 261679677
 336611124
 381259014
  73995607
 332416151
1453298236
1986152881
 388343933
 807477169
 861242803
1934274498
 345767925
 476279423
1924935738
1325558602
 825334243
1317635274
 134905925
1618044239
 631506869
 675019952
2001422005
  26308574
1995634011
1775484973
1562297091
1964678251
 314740901
1268570162
 710221486
1939894042
1514049229
1792643676
 915828956
1724390712
1477157980
    6563532
1647112540
1275212393
 394508488
1518927456
 556955271
 791108105
1844261189
1529208344

1427566642
1226045295
116806900
440123166
1919013627
427144072
1688325155
213976647
1636237860
1923410657
1549618281
182702660
2017694006
1057904683
1136744383
91411172
6303267
1194506154
1706542272
2094014020
1092982467
541312145
227619908
209424269
141605662
549142496
1693037207
2111632818
681013520
582992972
333507937
1414049255
733240153
1130230427
190493738
1768786018
1560397019
533141365
567616578
432304107
1215539156
180215082
1829270049
135287185
733685470
153044115
863006962
2024124455
338674142
247716735
1333626207
181198026
455368390
846495522
1988544667
79320989
260204515
746973287

1850603010
1015015810
1709502012
1275736390
1366305583
515236201
866859548
1813703217
1553004127
1589788265
667786859
1098071357
991502177
617098489
1565413112
916977454
404454593
280956916
306316401
1017171544
397679580
1813801718
2089470288
437786143
1202278604
706557017
931141112
1233047398
2071280957
1455040260
225641182
29222533
234945010
2142073064
1458617547
761078953
978251899
1607718701
1281907088
2083001175
1099532014
1371045894
1737857337
1172220670
1723292381
1655195863
341409463
2055753977
39581853
1453533614
483129542
467476811
516068878
965878699
128299071
985145439
619592575
1813627466

```
1273220553
1157745798
  26220005
 619558279
 536126172
1440331842
 121184505
 396451164
 797548811
 931571114
1471192250
 362690338
1706360386
1564802464
1790797553
 898350637
1279110304
 443053755
1139079825
 424828146
 875272429
1362312062
 947168028
2121235004
1506369355
1312461178
 404767434
1639284818
 614775439
 890426103
1546747636
1231485493
1884505518
1285050892
1057831785
 285229091
1532722664
  75342124
 507755637
Shell 290          > # SHELL TYPES * ( 1 + NUM FAILURES PER SHELL )
1254318178
2098139799
1666387071
 911875952
 351174903
 464973991
1770094307
1863956514
1102021507
 781766289
1124152402
1882617099
 812288517
 106729244
 245574656
 261665646
2033624739
1969836395
```

157793896
271130524
1546849755
651838142
1701564872
82724243
12483111
244906808
350621083
91152652
100794820
56264518
1917274567
1760483675
687736478
1920103439
255164492
1829412931
65802986
1189391847
136131375
986128304
94592367
501389520
1757408342
277662939
94653517
1324734924
1519183595
150068586
2142427089
339637926
2095730046
1299278160
1593726487
139772732
753384216
1368116222
702934821
315897538
1665507074
396096350
1307195449
1210629562
1990786372
324917526
246662309
527168661
1466374090
1612447373
390419262
738981513
1415473401
852828828
1594217849
1811552072
555113698
191420131

296809705
1343447161
1429251956
1795372280
249951116
126991137
2053654587
1671659035
1081266376
1467384084
1068948983
769812819
673887940
506079764
1166570195
908409930
1404052414
2060078470
260299136
1975677990
1119262381
1185081855
1108652391
581362415
1254318178
2098139799
1666387071
911875952
351174903
464973991
1770094307
1863956514
1102021507
781766289
1124152402
1882617099
812288517
106729244
245574656
261665646
2033624739
1969836395
157793896
271130524
1546849755
651838142
1701564872
82724243
12483111
244906808
350621083
91152652
100794820
56264518
1917274567
1760483675
687736478
1920103439

255164492
1829412931
65802986
1189391847
136131375
986128304
94592367
501389520
1757408342
277662939
94653517
1324734924
1519183595
150068586
2142427089
339637926
2095730046
1299278160
1593726487
139772732
753384216
1368116222
702934821
315897538
1665507074
396096350
1307195449
1210629562
1990786372
324917526
246662309
527168661
1466374090
1612447373
390419262
738981513
1415473401
852828828
1594217849
1811552072
555113698
191420131
296809705
1343447161
1429251956
1795372280
249951116
126991137
2053654587
1671659035
1081266376
1467384084
1068948983
769812819
673887940
506079764
78126602
1449793615

123408134
279851527
25918206
1313470009
1973272912
1363774876
1964472944
762430696
1120463904
2122378820
1351423507
773686062
1530940798
119025595
1053920743
539712780
364849192
906071962
9072619
1165804618
1601965645
1273904485
1013454024
407593237
1656650157
1478308453
1894682026
1087225688
193080365
320791323
302803943
133671222
1596438000
1242574742
1943290424
1774412320
894947055
283138655
900597425
1229537879
1796116157
807338923
168485917
224721985
823063902
96408747
1405245447
1510846185
2069670600
1666064138
1298966685
1923476645
951894902
1112045946
794837508
1457692859
343823458
169895195

```
1379116455
1818831623
1723360306
1351726752
1947015961
1864868926
 270058166
1223765756
  11633378
1773059583
1453891247
1671336745
 749608580
1203455213
 474053101
1038090897
1529368824
2124276371
 923038665
1924690854
 794844582
 910344442
 253656303
1634478624
 406225050
1693410375
1476957915
1136879695
 841790956
 731422712
1754526917
1621098442
1540070805
1102786008
1161994250
1337565556
 471351173
1657273749
 521902362
 962412522
ENDFILE.
```

LIST OF REFERENCES

1. CACI Products Company, MODSIM II, 3344 North Torrey Pines Court, La Jolla, CA 92037.

2. Naval Postgraduate School NPSOR-91-09, *Establishing Reliability Goals for Naval Major Caliber Ammunition*, by Michael P. Bailey, Marcelo Bartroli, Alexander Callahan, and Keebom Kang, March 1991.

3. Borland International, Inc., Turbo C and Turbo C++, 1800 Green Hills Road, P.O. BOX 66000, Scotts Valley, CA 95067.

4. Hughes, Wayne P. Jr., OA4602 Campaign Analysis Lecture, 7 July 1991, Naval Postgraduate School, Monterey, CA 93943.

5. Ozden, Mufit H., "Graphical Programming of Simulation Models in an Object-Oriented Environment," *Simulation*, volume 56 number 2, pp. 104-116, February 1991.

6. Hu, David, *An Object-Oriented Environment in C++*, Management Information Source, Inc., 1990.

7. Fishman, George S., *Principles of Discrete Event Simulation*, John Wiley & Sons, Inc., 1978.

8. CACI Products Company, *MODSIM II, The Language for Object-Oriented Programming, Reference Manual*, 1990 ed., 3344 North Torrey Pines Court, La Jolla, CA 92037.

9. Law, Averill M. and Kelton, David W., *Simulation Modeling and Analysis*, McGraw-Hill, 1982.

10. CACI Products Company, SIMGRAPHICS II, 3344 North Torrey Pines Court, La Jolla, CA 92037.

BIBLIOGRAPHY

Brantley, Paul, Fox, Bennet L., and Schrage, Linus E., *A Guide To Simulation,* Second Edition, Springer-Verlag New York, Inc., 1987.

Cox, Brad J., "There Is a Silver Bullet," *Byte*, volume 15, number 10, pp. 209-218, October 1990.

Gibson, Elizabeth, "Objects--Born and Bred," *Byte*, volume 15, number 10, pp. 245-254, October 1990.

Hughes, Wayne P. Jr., *Fleet Tactics,* Naval Institute Press, 1986.

Naval Warfare Publication, *Supporting Arms in Amphibious Operations*, NWP 22-2 (Rev. B).

Rice, John A., *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

Richardson, Doug, *Naval Armament*, Jane's Publishing Incorporated, 1982.

INITIAL DISTRIBUTION LIST

		copies
a)	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
b)	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
c)	LT Richard L. Darden Submarine Officer Advance Course Naval Submarine Base New London Groton, CT 06349	2
d)	Professor Michael P. Bailey, Code OR/BA Naval Postgraduate School Monterey, CA 93943-5000	5
e)	Professor Marcelo Bartroli, Code OR/BJ Naval Postgraduate School Monterey, CA 93943-5000	1
f)	LCDR Roger Stemp, Code OR/ST Naval Postgraduate School Monterey, CA 93943-5000	1
g)	John Bowden Naval Weapons Support Center Crane, IN 47522	1
h)	Hal Duncan CACI Products Company 3344 North Torrey Pines Court La Jolla, CA 92037	1